Secrecy by typing in the computational model

Stéphanie Delaune, Clément Herouard, Joseph Lallemand

Secrecy by typing in the computational model

Stéphanie Delaune
Univ. Rennes, CNRS, IRISA, France

Clément Hérouard
Univ. Rennes, CNRS, IRISA, France

Joseph Lallemand
Univ. Rennes, CNRS, IRISA, France

***Abstract*—In this paper, we propose a way to automate proofs of cryptographic protocols in the computational setting. We focus on non-deducibility – a weak notion of secrecy – and we aim to use type systems. Techniques based on typing were mainly used in symbolic models, and we show how they can be adapted to the CCSA framework to obtain computational guarantees.**

We consider for now a fixed set of primitives, namely symmetric and asymmetric encryption, as well as pairing (*i.e.* concatenation). Our approach has the usual benefit of type systems: it is modular, allows the security analysis for an unbounded number of sessions, and could be extended to other primitives (*e.g.* hashing) without excessive difficulties. We successfully applied our framework on several protocols from the literature and the ISO/IEC 11770 standard.

I. Introduction

Ensuring the absence of flaws in cryptographic protocols has become in the last few decades a major concern in the domain of security. Indeed, even small design errors in a protocol can lead to attacks with dramatic consequences.

Over the years, formal methods have proved to be an effective tool to ground the security of protocols on rigorous foundations, in order to obtain strong guarantees. Historically, two broad families of approaches have been explored: the *symbolic* and *computational* models. They differ in the level of abstraction they consider when modelling an attacker.

The symbolic approach, initiated by Dolev and Yao [18], represents the protocol by transition systems such as process algebras or multiset rewriting systems. The attacker is given control over the network, and the ability to manipulate messages in a set number of ways – decrypting a ciphertext if the key is known, *etc.* Notably, cryptographic primitives are assumed perfect: *e.g.* a ciphertext encrypted with a key that is secret leaks absolutely no information on the plaintext. This rather high level of abstraction allows for powerful automation procedures, which has lead to the development of efficient automated verification tools, *e.g.* ProVerif [9], [11], Tamarin [6], [26].

Computational models, on the other hand, forgo the ease of automation and reasoning, aiming instead for a greater level of precision. They model protocol participants and attackers as probabilistic polynomial time Turing Machines (PPTM), and only restrict the attacker by assuming he is unable to break specific assumptions on the cryptographic primitives involved. These assumptions are expressed as *cryptographic games*, *e.g.* Ind-Cpa, Ind-Cca, Int-Ctxt, *etc.* The computational model is closer to cryptographers' view on primitives, and offers strong guarantees, but is more difficult to automatically reason about: existing tools, notably CryptoVerif [10], EasyCrypt [5], are either limited in scope or non-automatic.

Recently, a new approach was proposed, called the Computationally Complete Symbolic Attacker (CCSA) [4]. In a way, it can be seen as a hybrid between the two families. It provides a symbolic framework, in which protocols and messages are represented by abstract terms. Security properties and axioms are written in a first-order logic, similarly to symbolic models, with the promise of allowing for relatively painless reasoning and automation. This symbolic representation is then related to a computational model, by interpreting the logic in a first-order model where abstract symbols are interpreted by PPTMs. This way, one can reason at the symbolic level and obtain computational guarantees. The CCSA framework has been implemented in a tool, Squirrel [2], [3], which is a proof assistant for this logic. It provides users with tactics that help cryptographic reasoning, but has for now a limited automation.

An important limitation of Squirrel has to do with non-deducibility properties. The tool's logic and tactics are well adapted to prove authentication, and equivalence properties, where one aims to show that two systems are indistinguishable to an attacker. This can express strong flavours of secrecy, *e.g.* stating that an attacker cannot distinguish the scenarios where protocol participants use the actual secret value or a random value. Squirrel is less convenient when considering weaker notions of secrecy, that are expressed as non-deducibility properties, typically stating that the attacker cannot obtain the exact value of a secret. Such properties are of course rather weak in themselves, but they can often be required as intermediate steps in more complex proofs. For instance, consider a protocol where two participants exchange some secret material x, and later feed it to a Key Derivation Function kdf to produce a key $k = \mathsf{kdf}(x)$. As part of a security proof for that protocol, one may need to show that the same key k could not have been derived earlier. Assuming that kdf is a Pseudo-Random Function, this can be proved if we can show that the same value x cannot have been previously deduced by the attacker. Note that the message x need not be strongly secret – it may for instance include public parts, such as transcripts from earlier messages. In contrast with indistinguishability proofs, proofs of non-deducibility tend to be rather tedious to write in Squirrel. In practice, one often has to prove strong secrecy instead, and obtain non-deducibility from it, which is inconvenient, and not always possible.

In this paper, we aim at designing techniques that improve Squirrel on the front of non-deducibility proofs, while

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

also improving the tool's automation. We propose a typing-based approach allowing to establish non-deducibility in the computational model by typechecking. By setting our work in the CCSA framework, we are able to implement it in the SQUIRREL prover.

Type systems are a common tool from the field of programming languages. They are used to construct sound (but often incomplete) procedures to statically ensure that a program is in some sense well-behaved, *i.e.* cannot raise certain runtime errors when executed. In the context of protocol analysis, type systems have been used notably in symbolic models, to establish trace properties such as non-deducibility and authentication (*e.g.* [1], [19]). Basically, the idea is to define types that express security guarantees on the messages to which they are associated – typically, a message's type could carry a level of secrecy and integrity. We show how to adapt such techniques from the symbolic to the CCSA model, to produce a computationally sound type system for non-deducibility. Note that showing the computational soundness of our approach requires reduction of cryptographic games expressing the assumptions on the primitives. The flexibility of a type system enables us to fine-tune the conditions of each typing rule individually, which helps in writing these reductions.

Related work. One somewhat related line of work, started by Laud in [23], focuses on secure information flow in a computational model, and studies non-interference properties. He uses that notion to study the security of programs using symmetric encryption. That work later led to a type system based on the same ideas [24]. While some ideas may be similar to our work on the surface, as noted *e.g.* in [27], the goal is rather different. Indeed, secure information flow aims at verifying programs, from the point of view of a single agent, to ensure they do not leak secrets, rather than verifying the protocol, meaning that communication on an attacker-controlled network is not modelled. The scope is also more limited: only programs that do not decrypt their data can be checked.

Another line of work focuses on the F* language. F* is a functional language with a very rich type system, notably supporting refinement types. Type checking then comes down to proving that the properties expressed by the types are satisfied. F* has been used to verify security protocols, in the computational model [12], [16], [17] as well as the symbolic one [8]. These approaches basically use F*'s types to write logical formulas encoding cryptographic games, or complex symbolic trace properties, and they use F* as a proof assistant to show those properties. In contrast, in our work, we aim for much simpler types, and a type system that is proved sound once and for all, so that in the end no proof effort is required from the user.

As mentioned earlier, typing techniques have been applied to protocol analysis in the symbolic model for some time, starting with the work of Abadi [1], based on ideas from information flow. A more recent example of a symbolic type system can be found in [19], which handles secrecy and authentication properties. In [15], the authors introduce a type system for symbolic equivalence. Here, we aim at obtaining computational guarantees, but we restrict our study to deducibility properties.

Finally, recently, another computational typing-based approach was proposed [20]. Like ours, it applies to reachability properties (not indistinguishability), and a limited set of primitives. The authors provide detailed proofs for symmetric encryption and hash functions, and only briefly mentioned how to deal with signature and asymmetric encryption. This work is more general than ours – we do not consider authentication properties, signatures and hash functions, and for technical reasons cannot handle key usability, *i.e.* exchanging a key, and then proving that a message encrypted with it remains secret. Gancher *et al.*'s work [20] does not suffer from these limitations. On the other hand, we provide detailed proofs for the primitives we considered (symmetric and asymmetric encryptions) and point out a shortcoming in their result regarding asymmetric encryption (see Section VII-C for a discussion regarding this point). Moreover, our approach has the advantage of being set in the CCSA model, and is integrated into SQUIRREL, while [20] is by essence a stand-alone tool. This integration in a more powerful proof assistant is crucial in our eyes. It offers the promise of interactions between SQUIRREL and our type system, *i.e.* automatically proving non-deducibility properties with our technique, and then using them as part of larger SQUIRREL proofs, that may *e.g.* involve indistinguishability and require user input. A more in-depth discussion of certain differences between this work and ours can be found in Section VII-C.

Contributions. In summary, we identify the following four contributions from this paper.

First, we propose a type system that can typecheck protocols expressed in the CCSA framework, in order to establish non-deducibility in the computational model. That system focuses on a symmetric encryption primitive. We prove the soundness of our type system, *i.e.* we show that by interacting with a well-typed protocol, no (active) attacker can obtain the value of a message whose type specifies it should be secret. This result holds even when considering unbounded numbers of sessions, and for protocols featuring persistent states.

Second, we illustrate the flexibility of our approach by extending it to handle asymmetric encryption – with minimal changes, only adding a few typing rules. We extend the soundness proof accordingly, which requires only local changes to account for the additional rules.

Third, we implement a type-checking procedure in the SQUIRREL tool, which, although not complete, can already type interesting examples.

Finally, we apply our approach to several examples, to show it is general enough to handle realistic case studies. We use it to show the non-deducibility of keys or exchanged secrets in several classic protocols from the literature, *e.g.* Needham-Schroeder, and Denning-Sacco, as well as protocols from the ISO/IEC 11770 standard for key management [21], [22].

II. Overview

In this section, we give an overview of our framework and tool, using as a running example the Wide Mouthed Frog (WMF) protocol [13]. Our implementation and all our case studies (including this running example) can be found in the supplementary material of this publication.

Example 1. *The* WMF *protocol, presented here without timestamps, is a key distribution protocol using randomised symmetric encryption and relying on a trusted server S.*

$$\begin{array}{ll} I \to S: & a, \{b, K\}^{ra}_{k_{as}} \\ S \to R: & \{a, K\}^{rs}_{k_{bs}} \end{array}$$

The agents a and b aim at authenticating each other, and establishing a session key K through the server S. The key k_{as} (resp. k_{bs}) is a long term key shared between a (resp. b) and the server.

```
channel c.
senc enc,dec.

abstract a:index→ message.

name Ks:index→ message.
name Ra:index*index*index→ message.
name Rs:index*index*index→ message.
name K:index*index*index→ message.

process Init(i:index,j:index,k:index)=
out(c,⟨ a(i),
       enc(⟨ a(j),K(i,j,k)⟩,Ra(i,j,k), Ks(i))⟩).

process Server(i:index,j:index,k:index)=
in(c, x);
if fst(x) = a(i)
   && dec(snd(x), Ks(i)) <> fail
   && fst(dec(snd(x), Ks(i))) = a(j) then
  let Key = snd(dec(snd(x), Ks(i))) in
  out(c, enc(⟨ fst(x), Key⟩, Rs(i,j,k), Ks(j))).

process Resp(i:index,j:index,k:index)= ...

system ( !i !j !k (
  Init(i,j,k)|Server(i,j,k)|Resp(i,j,k))).
```

Listing 1: WMF protocol in SQUIRREL

Listing 1 shows an excerpt from the formal description of the WMF protocol written in the input language of the SQUIRREL prover, which is close to the applied pi-calculus. We consider here a scenario involving only honest agents. The constant `a(i)` is used to model the i^{th} agent, and the long-term key shared between `a(i)` and the server is denoted by `Ks(i)`. The process `Init` represents the behaviour of the initiator of the protocol when played by agent `a(i)` with agent `a(j)`. This process can be executed an arbitrary number of times, and thus relies on an extra index `k`, which identifies the session. We also consider names `Ra`, and `Rs` to model the randomness used to produce ciphertexts. Those names are used only once and are thus parametrised by `i`, `j`, and `k`. WMF is a 3-party protocol, and is thus composed of three processes running in parallel. For instance, the server process `Server(i,j,k)` represents the role of the server whose purpose is to receive a message from agent `a(i)` and sends (after performing some tests) the corresponding answer to `a(j)`.

Internally, protocols in SQUIRREL are represented by *action systems*. In practice, the translation into action systems is performed automatically by the SQUIRREL tool from the applied pi-calculus specification. In the remainder of the paper, we state and prove our typing results considering protocols modelled directly as action systems. The formalism will be introduced in more detail in Section III. For now, we describe informally how the WMF protocol is translated into an action system. Roughly, we consider a set of actions representing the different steps of the protocol. For instance, I(`i`, `j`, `k`) is the action performed by the k^{th} session of agent `a(i)` playing the role of the initiator with agent `a(j)`; whereas S(`i`, `j`, `k`) represents the action of the server session `k` playing with agents `a(i)` and `a(j)` when the test succeeds on the message received as input. Actually there is also an action S'(`i`, `j`, `k`) representing the action of the server when the test fails, and similarly we can model the role of the responder.

Using the user syntax, we now express two secrecy properties on the WMF protocol. We first consider the secrecy of the session key as seen by the initiator.

```
goal secrecy_S : forall (tau:timestamp),
     forall (i,j,k:index),
    happens(tau) ⇒ att(frame@tau) <> K(i,j,k).
```

Listing 2: Secrecy goal (Initiator) in SQUIRREL

Here, frame is a macro which stands for the sequence of messages that have been emitted on the network so far. The predicate happens is used to ensure that the timepoint `tau` indeed happens in the trace under consideration, and att represents any computation that an attacker can perform. Roughly, our secrecy goal expresses that, for any trace in which the timepoint `tau` happens, the attacker is not able to compute `K(i,j,k)` for any `i`, `j`, `k`. This means that the keys `K(i,j,k)` are not deducible by the attacker, and expresses the secrecy of the session keys from the point of view of the server.

We may also express the secrecy of the session keys from the point of view of the initiator.

```
goal secrecy_I :
forall (tau:timestamp), forall (i,j,k:index),
happens(tau) && S(i,j,k)≤tau ⇒
att(frame@tau) <>
if cond@S(i,j,k) then Key@S(i,j,k) else Kfresh.
```

Listing 3: Secrecy goal (Initiator) in SQUIRREL

Here, cond is a macro which stands for the executability condition of action I(`i`, `j`, `k`), where the initiator recognises the input as a valid input message w.r.t. some pair of agents `a(i)` and `a(j)`. `Key` is also a macro that actually can be seen as syntactic sugar, and simply replaced by its value, here `snd(dec(snd(input@S(i,j,k)), Ks(i)))`.

In SQUIRREL, these two secrecy goals can be proved using a succession of tactics. Actually, as explained in introduction,

SQUIRREL is not well-suited to establish secrecy expressed as non-deducibility. A file showing how to prove these goals on a very simple scenario (only two agents and one session) with the current version of the SQUIRREL tool is available in the supplementary material of this publication, and contains ∼170 lines, of which ∼80 lines are proofs. The purpose of our typing result is to render this sort of proof shorter and more automatic, making things easier for the user.

The user has to declare the types of the names used in the protocol. Considering our running example, it means that the preamble will be slightly modified to insert typing information.

```
name Ks:index→message, SK[Cst a * High].
name Ra:index*index*index→message, Rand.
name Rs:index*index*index→message, Rand.
name K:index*index*index→message, High.
```

Listing 4: WMF protocol in SQUIRREL

The name K is given type High, which corresponds to terms for which we want to preserve the secrecy. The names Ra and Rs will be used as randomness when performing encryptions and are thus tagged Rand. Typically, relying on our type system, we will ensure that each random is used at most once, which is needed to apply some cryptographic assumptions (*e.g.* IND-CPA). Lastly, we have to give a type to the name Ks that is used as a key. Here, we give it type $\mathrm{SK}[\mathrm{Cst}_{\mathsf{a}}^{\infty} \times \mathrm{High}]$. Roughly, this expresses the fact that such a key is supposed to encrypt messages of type $\mathrm{Cst}_{\mathsf{a}}^{\infty} \times \mathrm{High}$ (pairs of an agent name and something not deducible by the attacker), and our type system will ensure that this is indeed the case. Once the types are provided, almost everything is done automatically. In fact, the proof scripts to establish these two secrecy goals are now very simple, and use very few tactics:

```
Proof. intro *. typing Meq. Qed.
Proof. intro *. expand cond. typing Meq. Qed.
```

Listing 5: WMF SQUIRREL proof script

The intro tactic allows us to introduce some hypotheses, whereas expand is used here to inline the content of the macro cond. More importantly, the proof scripts use the tactic typing whose design and soundness are the main purpose of this work. This is formally stated in Section IV, and more precisely Theorem 1. The use of this Theorem requires us to establish that the protocol under study is well-typed w.r.t. the types announced in the preamble. We have implemented a type checking procedure, which uses some heuristics, and is run automatically by the tool (see Section VI-A for more details).

III. BACKGROUND – THE CCSA LOGIC

Our work is set in the framework of the CCSA logic, introduced in [4], and generalised in [2], [3]. The CCSA logic consists of two layers, called *base logic* and *meta-logic*. Roughly speaking, the base logic expresses properties of messages, and the meta-logic expands it with a protocol model, allowing to reason about arbitrary executions.

A. Base logic

The base logic is a first-order logic, which deals with terms representing probabilistic polynomial time Turing Machines (PPTMs). We will use the notion of *negligible* probability: a function $f : \mathbb{N} \to [0;1]$ is negligible if it is asymptotically smaller than the inverse of any polynomial. We then write $f(\eta) \in \mathsf{negl}(\eta)$. Conversely, a function f is *overwhelming*, denoted by $f(\eta) \in \mathsf{ow}(\eta)$, when $1 - f$ is negligible.

1) Syntax: We assume disjoint sets $\mathcal{N}^{\mathcal{B}}$, $\mathcal{X}^{\mathcal{B}}$, $\mathcal{C}^{\mathcal{B}}$ of symbols. They are used respectively for names, *i.e.* random values generated by protocol agents, variables, and constants. The sets $\mathcal{N}^{\mathcal{B}}$ and $\mathcal{C}^{\mathcal{B}}$ are assumed finite. We consider a *function signature*, *i.e.* a finite set of function symbols $\mathcal{F}^{\mathcal{B}}$. We assume that $\mathcal{F}^{\mathcal{B}}$ contains (among others) the following set $\mathcal{F}_0$ of symbols (with their arity) representing a few usual operations:

- pairing and projections $\langle \cdot, \cdot \rangle/2$, $\mathsf{fst}/1$, $\mathsf{snd}/1$,
- equality $\cdot = \cdot/2$,
- conditional branching $\mathsf{if} \cdot \mathsf{then} \cdot \mathsf{else} \cdot /3$,
- symmetric encryption and decryption $\mathsf{senc}/3$, $\mathsf{sdec}/2$,
- $\mathsf{zeros}/1$ constructs a string of 0s of a given length.

We assume constants $\mathsf{true}, \mathsf{false}, \mathsf{empty}, \mathsf{fail}$ in $\mathcal{C}^{\mathcal{B}}$. We further assume $\mathcal{C}^{\mathcal{B}}$ is partitioned into $\mathcal{C}^{\mathcal{B},0} \cup \bigcup_{\mathsf{c} \in \mathcal{C}} \mathcal{C}_c^{\mathcal{B},\infty}$, where $\mathcal{C}$ is a finite set of identifiers. This partition will be convenient when later on, in the meta-logic, we will consider replicated constants, designated by the same symbol. They are used *e.g.* to represent agent names. Note that the $\mathsf{if} \cdot \mathsf{then} \cdot \mathsf{else} \cdot /3$ function symbol allows us to define other boolean connectives, *e.g.* $\mathsf{if}\ u\ \mathsf{then}\ v\ \mathsf{else}\ \mathsf{false}$ for $u \wedge v$.

These symbols will later on be interpreted as deterministic computations. In order to model randomised encryption, the encryption function senc must therefore take a random value as argument. Rather than using a name from $\mathcal{N}^{\mathcal{B}}$ to represent that value, we assume an additional finite set $\mathcal{R}^{\mathcal{B}}$ of symbols for such encryption randomness. As we will see later on, tracking randomness usage will be crucial in order for our cryptographic reasoning to be sound, and using a distinct set of symbols will help make such conditions more legible.

Messages are modelled by terms constructed over $\mathcal{N}^{\mathcal{B}}$, $\mathcal{X}^{\mathcal{B}}$, $\mathcal{C}^{\mathcal{B}}$, $\mathcal{R}^{\mathcal{B}}$, $\mathcal{F}^{\mathcal{B}}$, and a special unary symbol att, which will represent arbitrary computations by the attacker.

2) Semantics: Base logic terms and formulas are interpreted in a class of first-order models called *computational models*. A computational model $\mathbb{M}$ interprets terms into the domain of PPTMs that have read-only access to a pair $\rho = (\rho_{\mathsf{h}}, \rho_{\mathsf{a}})$ of (infinite) random tapes, and run in polynomial time w.r.t. a security parameter η (provided to them in unary). Random tapes ρ_{h} and ρ_{a} represent the random values drawn respectively by protocol agents and the attacker, and the security parameter η represents the length of encryption keys used. Each function and constant symbol f is interpreted as a deterministic, independent of ρ, PPTM $[\![\mathsf{f}]\!]^{\mathbb{M}}$, that receives as inputs 1^{η} and its arguments.

We write $[\![t]\!]^{\mathbb{M}}_{\sigma}$ the interpretation of term t in model $\mathbb{M}$ with substitution σ, where σ is a mapping from the variables in t to PPTMs in the domain of $\mathbb{M}$. It is defined inductively as expected: variables in $\mathcal{X}^{\mathcal{B}}$ are interpreted by σ, names and

random values in $\mathcal{N}^{\mathcal{B}}$, $\mathcal{R}^{\mathcal{B}}$ are interpreted as machines reading a random string of length η in ρ_{h} (in such a way that different names are associated with disjoint areas of the tape). When t is a ground term, we may omit σ and simply write $[\![t]\!]^{\mathbb{M}}$.

We restrict the models $\mathbb{M}$ we consider by requiring that all function symbols in $\mathcal{F}^{\mathcal{B}}$, as well as constants in $\mathcal{C}^{\mathcal{B}}$, are interpreted as PPTMs that do not access the random tape ρ. We further require that:

- the function symbols for conditional branching, equality, zero, and the default constants are interpreted as expected;
- pairing and projections are interpreted in such a way that for any σ, t_1, t_2, η, ρ:
$$[\![\mathsf{fst}(\langle t_1, t_2\rangle)]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho) = [\![t_1]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho),$$
and similarly for snd;
- senc and sdec satisfy the IND-CPA and INT-CTXT assumptions (recalled in Appendix A-A), and represent a correct encryption scheme, *i.e.* for any m, k, r, η, ρ,
$$[\![\mathsf{sdec}(\mathsf{senc}(m, k, r), k)]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho) = [\![m]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho).$$
- two distinct constant symbols $\mathsf{c}, \mathsf{c}' \in \mathcal{C}^{\mathcal{B}}$ are always interpreted by distinct values: for all η, $[\![\mathsf{c}]\!]^{\mathbb{M}}(1^\eta) \neq [\![\mathsf{c}']\!]^{\mathbb{M}}(1^\eta)$.

The att symbol may be interpreted as any PPTM that only accesses the ρ_{a} tape. Note that with these restrictions, for any term t, the length of the bitstring $[\![t]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho)$ is bounded (for all ρ) by a polynomial in η.

Then, satisfiability is expressed as overwhelming truth. Given a base logic term ϕ, we say that ϕ is satisfied in $\mathbb{M}$, σ if:

$$\mathbb{P}_{\rho}\left[[\![\phi]\!]^{\mathbb{M}}_{\sigma}(1^\eta, \rho) = [\![\mathsf{true}]\!]^{\mathbb{M}}_{\sigma}(1^\eta))\right] \in \mathsf{ow}(\eta).$$

B. Meta-logic

In order to study properties of protocols, [2] introduces the meta-logic level. Basically, a protocol execution model is introduced in the form of an *action system*. Meta-logic terms can contain macros that refer to messages produced in any action along the execution, *e.g.* $\mathsf{output}@\tau$ refers to the message output at time τ. Using these macros, meta-logic formulas can then express generic properties on protocol executions.

In order to allow reasoning on many sessions of protocols, the meta-logic introduces a notion of *indices*. Constants, names, and encryption randomness may take as arguments indices, that identify the protocol session they belong to. For sake of clarity, we make the choice to present a version of the meta-logic that does not feature states. As already stated in introduction, our typing result has actually been established for stateful protocols, and the interested reader will find the full presentation of the meta-logic, as well as the typing result and its proof in this more general setting in Appendix.

1) Syntax: The syntax of the meta-logic consists of terms of three sorts, namely *index*, *timestamp* and *message*. Formally, we assume infinite sets $\mathcal{I}$, $\mathcal{X}$, $\mathcal{T}$ of index, message, and timestamp variables. We assume two symbols init, pred, used to represent the initial timestamp and a predecessor operation on timestamps. We also assume finite sets $\mathcal{N}$, $\mathcal{R}$, $\mathcal{C}$ of *indexed* names, random values, and constants: each of these symbols has an index arity. We denote by $\mathcal{C}^0$ (resp. $\mathcal{C}^\infty$) the subset of $\mathcal{C}$ of index arity 0 (resp. non-zero). We assume a finite set of function symbols $\mathcal{F}$ of index arity 0, but each symbol has a message arity $k \geq 0$. In our formal development, to distinguish index arguments from message arguments, we write them between brackets, *e.g.* $\mathsf{n}[i_1, \ldots, i_k]$ vs $\mathsf{f}(m_1, \ldots, m_l)$. We consider the set

$$\mathcal{M}_0 = \{\mathsf{output}, \mathsf{input}, \mathsf{frame}, \mathsf{exec}, \mathsf{cond}\}$$

containing five *macro* symbols (of index arity 0), and finally, we assume a finite set $\mathcal{A}$ of *action* symbols, given with their index arity, used to denote protocol steps.

A macro is identified by its name m_0 and will be evaluated at a particular timestamp τ: informally, $\mathsf{m}_0@\tau$ refers to the value of m_0 at point τ in the execution. The macros $\mathsf{input}@\tau$ and $\mathsf{output}@\tau$ denote the messages input and output at timepoint τ in the execution trace; and $\mathsf{frame}@\tau$ is roughly the sequence of all messages output by any agent during the execution up to timestamp τ. The frame is crucial for expressing security properties, as it represents the full list of messages seen by the attacker. Lastly, $\mathsf{cond}@\tau$ represents the condition for executing the output at timepoint τ, whereas $\mathsf{exec}@\tau$ can be seen as the executability condition to reach timepoint τ, *i.e.* the conjunction of all the conditions occurring before τ.

The only terms of sort index are index variables. Terms of sort timestamp and message are built as follows:

$$\begin{array}{rcl} T & ::= & \tau \mid \mathsf{A}[\vec{i}] \mid \mathsf{init} \mid \mathsf{pred}(T) \\ t & ::= & x \mid \mathsf{m}_0@T \mid \mathsf{n}[\vec{i}] \mid \mathsf{f}(\vec{t}) \end{array}$$

where $\tau \in \mathcal{T}$, $\mathsf{A} \in \mathcal{A}$, $x \in \mathcal{X}$, $\mathsf{m}_0 \in \mathcal{M}_0$, $\mathsf{n} \in \mathcal{N} \cup \mathcal{R} \cup \mathcal{C}$, $\mathsf{f} \in \mathcal{F}$, $\vec{i}$ a vector of indices, and $\vec{t}$ a vector of messages. We sometimes call *meta-terms* the meta-logic terms.

An *action* A is defined as:

$$\mathsf{A}[\vec{i}].(\phi_{\mathsf{A}[\vec{i}]}, o_{\mathsf{A}[\vec{i}]})$$

where $\vec{i}$ is a vector of distinct indices in $\mathcal{I}$, understood in this notation as bound variables, $\phi_{\mathsf{A}[\vec{i}]}$, and $o_{\mathsf{A}[\vec{i}]}$ are messages called the *condition* and *output*.

A *protocol* $\mathcal{P} = (\mathcal{A}\mathsf{ct}, \prec)$ is composed of:

- a finite set $\mathcal{A}\mathsf{ct}$ of actions (one for each symbol in $\mathcal{A}$);
- and a partial order $\prec$ on terms of the form $\mathsf{A}[\vec{i}]$. $\prec$ must be invariant by alpha-renaming indices.

We require that $\mathcal{A}\mathsf{ct}$ contains a particular action init, with no indices, that is smaller than all others for $\prec$. Its condition is $\phi_{\mathsf{init}} = \mathsf{true}$, its output $o_{\mathsf{init}} = \mathsf{empty}$.

In addition, we require that in action $\mathsf{A}[\vec{i}]$, all timestamps refer to earlier actions:

- either $\mathsf{A}'[\vec{j}]$ such that $\mathsf{A}'[\vec{j}] \prec \mathsf{A}[\vec{i}]$;
- or $\mathsf{pred}(\mathsf{A}[\vec{i}])$ when $\mathsf{A} \neq \mathsf{init}$;
- or $\mathsf{A}[\vec{i}]$ itself as a parameter to input.

Moreover, we require that all randoms $\mathsf{r} \in \mathcal{R}$ appearing in $\mathsf{A}[\vec{i}]$ are applied to exactly $\vec{i}$, *i.e.* $\mathsf{r}[\vec{i}]$.

Example 2. *The protocol introduced in Example 1 can be modelled in our formalism as* $\mathcal{P}_{\mathsf{WMF}} = (\mathcal{A}\mathsf{ct}, \emptyset)$ *where* $\mathcal{A}\mathsf{ct}$ *contains five actions, namely* $\mathsf{S}[i, j, k]$, $\mathsf{I}[i, j, k]$, *and* $\mathsf{R}[i, j, k]$,

as well as the special init *action, and an extra action modelling the else branch of the server role. We detail below the action modelling the server when the condition is satisfied. Intuitively* $\mathsf{S}[i,j,k]$ *is the action performed by the server when receiving a message from agent* $\mathsf{a}[i]$ *to communicate with agent* $\mathsf{a}[j]$*. The index* k *allows us to model that this action can be executed many times:*

$$\begin{array}{rl} \phi_{\mathsf{S}[i,j,k]} = & \mathsf{fst}(\mathsf{input}@\mathsf{S}[i,j,k]) = \mathsf{a}[i] \\ & \wedge\ \neg(t^{\mathsf{S}}_{\mathsf{sdec}} = \mathsf{fail}) \ \wedge\ \mathsf{fst}(t^{\mathsf{S}}_{\mathsf{sdec}}) = \mathsf{a}[j] \\ o_{\mathsf{S}[i,j,k]} = & \mathsf{senc}(\langle \mathsf{a}[i], \mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})\rangle, \mathsf{k_s}[j], \mathsf{r_s}[i,j,k]) \end{array}$$

where $t^{\mathsf{S}}_{\mathsf{sdec}} \mathrel{\widehat{=}} \mathsf{sdec}(\mathsf{snd}(\mathsf{input}@\mathsf{S}[i,j,k]), \mathsf{k_s}[i])$.

We write $\neg b$ *as a shortcut for* if b then false else true*, and similarly for* $b_1 \wedge b_2$*.*

2) Semantics: The semantics of the meta-logic is defined by translating meta-logic terms into base-logic terms, and rely on the notion of trace model. Given a protocol $\mathcal{P}$, a *trace model* $\mathbb{T}$ of $\mathcal{P}$ is composed of

- a finite index domain $\mathcal{D}_\mathcal{I} \subseteq \mathbb{N}$;
- a timestamp domain $\mathcal{D}_\mathcal{T}$ containing undef, and elements of the form $\mathsf{A}[\vec{k}]$ where $\mathsf{A} \in \mathscr{A}$ and $\vec{k} \in \mathcal{D}_\mathcal{I}$;
- a total ordering $<_\mathcal{T}$ on $\mathcal{D}_\mathcal{T} \setminus \{\mathsf{undef}\}$, compatible with $\prec$ in the sense that, for any $\sigma : \mathcal{I} \to \mathcal{D}_\mathcal{I}$,
$$\mathsf{A}[\vec{i}] \prec \mathsf{A}'[\vec{j}] \text{ implies } \mathsf{A}[\sigma(\vec{i})] <_\mathcal{T} \mathsf{A}'[\sigma(\vec{j})];$$
- $\sigma_\mathcal{I} : \mathcal{I} \to \mathcal{D}_\mathcal{I}$, $\sigma_\mathcal{T} : \mathcal{T} \to \mathcal{D}_\mathcal{T}$ are mappings interpreting index and timestamp variables.

For a trace model $\mathbb{T}$, we define a predecessor function $\mathsf{pred}_\mathcal{T}$ as expected in accordance with $<_\mathcal{T}$, and we set $\mathsf{pred}_\mathcal{T}(\mathsf{init})$ to undef. Let $\mathbb{T}\{\tau \mapsto T\}$ denote the trace model $\mathbb{T}$ where $\sigma_\mathcal{T}$ is updated to map τ to T, and similarly for an update of $\sigma_\mathcal{I}$.

Example 3. *Consider the protocol* $\mathcal{P}_{\mathsf{WMF}}$ *introduced in Example 2. A possible trace model* $\mathbb{T}$ *associated to this protocol is the tuple* $(\mathcal{D}_\mathcal{I}, \mathcal{D}_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$ *where:*

- $\mathcal{D}_\mathcal{I} = \{1,2,3,4,5\}$*;*
- $\mathcal{D}_\mathcal{T} = \{\mathsf{undef}, \mathsf{init}, \mathsf{S}[1,2,1], \mathsf{S}[1,2,3], \mathsf{R}[1,2,4]\}$*;*
- $<_\mathcal{T}$ *is a total ordering such that:*
  $\mathsf{init} <_\mathcal{T} \mathsf{S}[1,2,1] <_\mathcal{T} \mathsf{S}[1,2,3] <_\mathcal{T} \mathsf{R}[1,2,4]$
- $\sigma_\mathcal{I}$ *and* $\sigma_\mathcal{T}$ *are mappings that interpret index and timestamp variables to* $\mathcal{D}_\mathcal{I}$ *and* $\mathcal{D}_\mathcal{T}$ *respectively. For illustration purposes, let* $\sigma_\mathcal{I}(i) = \sigma_\mathcal{I}(k) = 1$*, and* $\sigma_\mathcal{I}(j) = 2$*.*

In this case, we have $\mathsf{pred}_\mathcal{T}(\mathsf{S}[1,2,3]) = \mathsf{S}[1,2,1]$.

Given a protocol $\mathcal{P}$ and a trace model $\mathbb{T}$, we now define the translation from meta-logic terms to base-logic terms. We take an instance of the base logic such that $\mathcal{F}^\mathcal{B} = \mathcal{F}$, $\mathcal{N}^\mathcal{B} = \{\mathsf{n}_{\vec{k}} \mid \mathsf{n} \in \mathcal{N}, \vec{k} \in \mathcal{D}_\mathcal{I}\}$, and similarly for randoms $\mathcal{R}^\mathcal{B}$. We take $\mathscr{C}^\mathcal{B} = \mathscr{C}^0 \cup \bigcup_{c \in \mathscr{C}^\infty} \{\mathsf{c}_{\vec{k}} \mid \vec{k} \in \mathcal{D}_\mathcal{I}\}$, and we choose $\mathcal{X}^\mathcal{B} = \mathcal{X} \cup \mathcal{X}_{\mathscr{M}_0}$, where $\mathcal{X}_{\mathscr{M}_0} = \{x_{\mathsf{m}_0@T} \mid \mathsf{m}_0 \in \mathscr{M}_0, T \in \mathcal{D}_\mathcal{T}\}$ is a set of variables that will be used as stand-ins for macros.

The translation of terms of sort timestamp is as expected, using $\sigma_\mathcal{T}$, $\sigma_\mathcal{I}$ and $\mathsf{pred}_\mathcal{T}$:

$$\overline{\tau}^\mathbb{T} = \sigma_\mathcal{T}(\tau), \quad \overline{\mathsf{A}[\vec{i}]}^\mathbb{T} = \mathsf{A}[\sigma_\mathcal{I}(\vec{i})], \quad \overline{\mathsf{pred}(T)}^\mathbb{T} = \mathsf{pred}_\mathcal{T}(\overline{T}^\mathbb{T}).$$

For terms t of sort message, we proceed in two steps. First, we define $\overline{t}^\mathbb{T}$ to be t where each macro $\mathsf{m}_0@T$ is replaced with the variable $x_{\mathsf{m}_0@T}$ that will be instantiated later with the body of the macro, and each meta-logic construct (*i.e.* function symbols, and names) is translated using its counterpart in the base logic. As a second step, we define how macro variables are instantiated. Actually, we simultaneously define $\theta^\mathbb{T}_\mathcal{P}$, and $(\cdot)^\mathbb{T}_\mathcal{P}$ allowing us to respectively interpret macros, and more generally translate any meta-term into a base-logic term (once the trace model $\mathbb{T}$ is fixed).

Formally, for any meta-term t and trace model $\mathbb{T}$, $(t)^\mathbb{T}_\mathcal{P}$ is $\overline{t}^\mathbb{T}\theta^\mathbb{T}_\mathcal{P}$, and for any $\mathsf{m}_0 \in \mathscr{M}_0$, $T \in \mathcal{D}_\mathcal{T}$:

- when $T = \mathsf{undef}$, $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{m}_0@T})$ is false if $\mathsf{m} \in \{\mathsf{cond}, \mathsf{exec}\}$ and empty otherwise;
- when $T = \mathsf{A}[\vec{k}]$ for an action
$$\mathsf{A}[\vec{i}].(\phi_{\mathsf{A}[\vec{i}]}, o_{\mathsf{A}[\vec{i}]}),$$
denoting $\mathbb{T}' = \mathbb{T}\{\vec{i} \mapsto \vec{k}\}$, we let
$$\begin{array}{ll} -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{output}@T}) & = (\text{if } \phi_{\mathsf{A}[\vec{i}]} \text{then } o_{\mathsf{A}[\vec{i}]} \text{ else empty})^{\mathbb{T}'}_\mathcal{P} \\ -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{cond}@T}) & = (\phi_{\mathsf{A}[\vec{i}]})^{\mathbb{T}'}_\mathcal{P} \end{array}$$
If A is init, we let $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{input}@T}) = \theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{frame}@T}) = \mathsf{empty}$, and $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{exec}@T}) = \mathsf{true}$. Otherwise:
$$\begin{array}{ll} -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{input}@T}) & = (\mathsf{att}(\mathsf{frame}@\mathsf{pred}(\mathsf{A}[\vec{i}])))^{\mathbb{T}'}_\mathcal{P} \\ -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{frame}@T}) & = (\langle \mathsf{exec}@\mathsf{A}[\vec{i}], \\ & \qquad \langle \text{if } \mathsf{exec}@\mathsf{A}[\vec{i}] \text{then } \mathsf{output}@\mathsf{A}[\vec{i}] \\ & \qquad \text{else empty}, \mathsf{frame}@\mathsf{pred}(\mathsf{A}[\vec{i}])\rangle\rangle)^{\mathbb{T}'}_\mathcal{P} \\ -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{exec}@T}) & = (\mathsf{cond}@T \wedge \mathsf{exec}@\mathsf{pred}(\mathsf{A}[\vec{i}]))^{\mathbb{T}'}_\mathcal{P}. \end{array}$$

As we can see, the output macro is replaced with the meta-term (under its conditional) as specified by the protocol, and then interpreted. The cond is handled similarly, and produces a base logic formula corresponding to the condition of the action. The macro exec is translated as the conjunction of all the past conditions. The translation of frame gathers all the information available to the attacker at some execution point (*i.e.* all the outputs emitted so far, but also the executability condition at each step). Finally, as the attacker controls the network, the input macro is interpreted using the attacker symbol att: it can be any computation that the attacker could perform on $\mathsf{frame}@\tau$, *i.e.* the messages seen so far. Compared to [3], this translation is very similar, except that the expansion of the output macro explicitly involves the condition $\phi_{\mathsf{A}[\vec{i}]}$. This makes no difference when considering the frame macro, as that condition was already present there, but ensures we only consider the output of an action when that action can effectively be performed, which may provide useful information when typechecking.

Example 4. *Going back to our running example, we have* $\overline{\mathsf{pred}(\mathsf{S}[i,j,k])}^\mathbb{T} = \mathsf{pred}_\mathcal{T}(\mathsf{S}[1,2,1]) = \mathsf{init}$*. Now, considering the meta-term* $o_{\mathsf{S}[i,j,k]}$*, we have:*

$$\overline{o_{\mathsf{S}[i,j,k]}}^\mathbb{T} = \mathsf{senc}(\langle \mathsf{a}_{(1)}, \mathsf{snd}(\overline{t_{\mathsf{sdec}}}^\mathbb{T})\rangle, \mathsf{k_s}_{(2)}, \mathsf{r_s}_{(1,2,1)})$$

where $\overline{t^{\mathsf{S}}_{\mathsf{sdec}}}^{\mathbb{T}} \mathrel{\widehat{=}} \mathsf{sdec}(\mathsf{snd}(x_{\mathsf{input@S}[1,2,1]}), \mathsf{k}_{\mathsf{s}(1)})$. *Then, we have* $(o_{\mathsf{S}[i,j,k]})^{\mathbb{T}}_{\mathcal{P}} = \overline{o_{\mathsf{S}[i,j,k]}}^{\mathbb{T}}\theta$ *where* $\theta(x_{\mathsf{input@S}[1,2,1]}) = \mathsf{att}(\mathsf{empty})$.

Given a trace model $\mathbb{T}$ of a protocol $\mathcal{P}$, and a term T of sort timestamp, we say that T *happens* in $\mathbb{T}$ if $\overline{T}^{\mathbb{T}} \neq \mathsf{undef}$. Finally, we introduce a notion of secrecy expressed as non-deducibility from messages outputted at any point in time.

Definition 1. *Let $\mathcal{P}$ be a protocol and t a ground meta-logic term. Let τ be an arbitrary fresh timestamp variable.*

We say that t is secret *in $\mathcal{P}$ if for any trace model $\mathbb{T}$ of $\mathcal{P}$ in which τ as well as all timestamps occurring in t happen, and for any computational model $\mathbb{M}$:*

$$(\mathsf{att}(\mathsf{frame@}\tau) \neq t)^{\mathbb{T}}_{\mathcal{P}} \text{ is satisfied in } \mathbb{M}.$$

In SQUIRREL, a predicate $\mathsf{happens}(\tau)$ is provided in the logic, allowing us to rewrite this secrecy notion as a formula.

IV. Main result

In this section, we present our main contribution: a type system to prove secrecy in the CCSA model. For the sake of clarity, we present this result for the notion of protocols introduced in Section III, *i.e.* protocols that do not feature states. Actually, we have proved this result in a more general setting – it applies to stateful protocols as well, with some minor changes. These changes are formally presented in Appendix B. The reader does not need to understand how stateful protocols are modelled in our framework to understand our typing result and the proof sketch presented in Section V. However, the formal proofs detailed in Appendices C, D, and E are written in the general setting, and thus consider protocols that may involve states.

A. Types and environment

The grammar for *message types* is as follows:

$$\mathrm{T} ::= \mathrm{Bool} \mid \mathrm{Cst}^0_{\mathsf{c}} \mid \mathrm{Cst}^\infty_{\mathsf{c}'} \mid \mathrm{Msg} \mid \mathrm{Low} \mid \mathrm{High} \mid \mathrm{T} + \mathrm{T} \mid \mathrm{T} \times \mathrm{T}$$

where $\mathsf{c} \in \mathscr{C}^0$ and $\mathsf{c}' \in \mathscr{C}^\infty$.

Type Bool is for boolean values. We make a distinction between constants of index arity 0 and non-zero: type $\mathrm{Cst}^0_{\mathsf{c}}$'s single inhabitant is the constant c, while type $\mathrm{Cst}^\infty_{\mathsf{c}}$ is given to all instances of $\mathsf{c}[\vec{i}]$. We write $\mathrm{Cst}^\star_{\mathsf{c}}$ to represent either $\mathrm{Cst}^0_{\mathsf{c}}$ or $\mathrm{Cst}^\infty_{\mathsf{c}}$, in contexts where the distinction is unimportant. Type Msg is the least precise type. Type Low is given to public messages, whereas High is used for secret ones. Our system also allows for more complex types, namely sum and product types, constructed over base types. Type $\mathrm{T}_1 + \mathrm{T}_2$ designates a message that can be of type either T_1 or T_2 depending on the execution, and type $\mathrm{T}_1 \times \mathrm{T}_2$ is for pairs of messages of types T_1 and T_2.

In addition, we define special *key types*, that are given to symmetric keys. They are of the form $\mathrm{SK}[\mathrm{T}]$, where T is a message type denoting the type of plaintexts. Note that key types are not themselves message types.

The type connectors $+$ and $\times$ are not commutative or associative. Later on, for simplicity, we will write $\mathrm{T}_1 + \cdots + \mathrm{T}_n$ for $\mathrm{T}_1 + (\mathrm{T}_2 + (\cdots + \mathrm{T}_n)\dots)$, and similarly for $\times$.

We define a subtyping relation $\mathrm{T} \leq \mathrm{T}'$ over message types, which as usual allows a value of type T to be given type T'. Formally, $\leq$ is the least preorder such that:

- $\mathrm{High} \times \mathrm{Msg},\ \mathrm{Msg} \times \mathrm{High} \leq \mathrm{High}$;
- $\mathrm{Bool},\ \mathrm{Cst}^\star_{\mathsf{c}},\ \mathrm{Low} \times \mathrm{Low} \leq \mathrm{Low}$ for any c;
- $\mathrm{Cst}^0_{\mathsf{false}},\ \mathrm{Cst}^0_{\mathsf{true}} \leq \mathrm{Bool}$;
- $\mathrm{T} \leq \mathrm{Msg}$ for all T;
- $\mathrm{T}_1,\ \mathrm{T}_2 \leq \mathrm{T}_1 + \mathrm{T}_2$;
- $\mathrm{T}_1 \times \mathrm{T}_2 \leq \mathrm{T}'_1 \times \mathrm{T}'_2$ when $\mathrm{T}_1 \leq \mathrm{T}'_1$ and $\mathrm{T}_2 \leq \mathrm{T}'_2$.

When typechecking terms, we store the types of keys, variables, and macro symbols in a *typing environment*.

Definition 2. *A* typing environment $(\Gamma; R)$ *is composed of*

- *a finite set of bindings Γ giving a message or key type T to some elements of $\mathcal{X} \cup \mathcal{N}$;*
- *a finite set $R \subseteq \mathcal{R}$ of random symbols that can be used for encryptions.*

We say that Γ, and by extension $(\Gamma; R)$, is well-formed *if*

(i) Γ does not contain multiple bindings for the same symbol;
(ii) for $n \in \mathcal{N}$, $\Gamma(n)$ is either High, Low or a key type;
(iii) for $x \in \mathcal{X}$, $\Gamma(x)$ is a message type.

We write $R \sqcup R'$ to denote the disjoint union of R and R', *i.e.* $R \cup R'$ with the assertion that R and R' are disjoint.

Example 5. *Consider the typing environment $(\Gamma_0; R_0)$ where $R_0 = \{\mathsf{r_a}, \mathsf{r_s}\}$, and $\Gamma_0 = \{\mathsf{k}, \mathsf{k_{fresh}} : \mathrm{High},\ \mathsf{k_s} : \mathrm{SK}[\mathrm{Msg}]\}$. The environment $(\Gamma_0; R_0)$ is well-formed: names k, $\mathsf{k_{fresh}}$ are bound to High, and $\mathsf{k_s}$ is bound to the key type $\mathrm{SK}[\mathrm{Msg}]$. This environment does not contain variables.*

B. Type system

Our type system produces *typing judgements*, *i.e.* expressions of the form $\Gamma; R \vdash t : \mathrm{T}$ where $(\Gamma; R)$ is a well-formed typing environment, t a meta-term, and T a message type.

The type system is composed of typing rules, which can be used to derive typing judgements. The typing rules are sorted into three categories.

1) Figure 3 displays the rules for typing meta-terms that are actually macros. These rules are mostly straightforward, as the type of such a term is entirely determined by the macro symbol.
2) Figures 1 and 2, display the rules for typing meta-terms that do not contain any macro. We split them into two parts to help structure the soundness proof in the next section. Indeed, we first establish the soundness of the type system composed only of the rules in Figure 1, and then add the rules in Figure 2.

Most of the rules in Figure 1 are rather intuitive. Rule NAME allows to type names as indicated in the typing environment. Rule SUB-TYPING allows to sub-type terms as expected, whereas rules CST-0 and CST-∞ give the corresponding constant type to constants. Rules FUN-LOW and FUN-MSG deal with the case where an arbitrary function symbol is applied to public terms or terms with the generic Msg type. Other rules apply to some specific function symbols. Rule PAIR gives a pair type as expected. Rule

$$\frac{\Gamma(\mathsf{n}) = \mathrm{T}}{\Gamma; R \vdash \mathsf{n}[\vec{i}] : \mathrm{T}}\ \textsc{Name} \qquad \frac{\Gamma; R \vdash t : \mathrm{Msg}}{\Gamma; R \vdash \mathsf{zeros}(t) : \mathrm{Low}}\ \textsc{Zeros}$$

$$\frac{\Gamma; R_1 \vdash t_1 : \mathrm{T}_1 \qquad \Gamma; R_2 \vdash t_2 : \mathrm{T}_2}{\Gamma; R_1 \sqcup R_2 \vdash \langle t_1, t_2 \rangle : \mathrm{T}_1 \times \mathrm{T}_2}\ \textsc{Pair}$$

$$\frac{\mathsf{c} \in \mathscr{C}^0}{\Gamma, R \vdash \mathsf{c} : \mathrm{Cst}^0_{\mathsf{c}}}\ \textsc{Cst-0} \qquad \frac{\mathsf{c} \in \mathscr{C}^\infty}{\Gamma, R \vdash \mathsf{c}[\vec{i}] : \mathrm{Cst}^\infty_{\mathsf{c}}}\ \textsc{Cst-}\infty$$

$$\frac{\Gamma; R \vdash t : \mathrm{T} \quad \mathrm{T} \leq \mathrm{T}'}{\Gamma; R \vdash t : \mathrm{T}'}\ \textsc{Sub-Typing}$$

$$\frac{\Gamma; R_i \vdash t_i : \mathrm{Low} \quad \text{for } i = 1, \ldots, n}{\Gamma; R_1 \sqcup \ldots \sqcup R_n \vdash \mathsf{f}(t_1, ..., t_n) : \mathrm{Low}}\ \textsc{Fun-Low}$$

$$\frac{\Gamma; R_i \vdash t_i : \mathrm{Msg} \quad \text{for } i = 1, \ldots, n}{\Gamma; R_1 \sqcup \ldots \sqcup R_n \vdash \mathsf{f}(t_1, ..., t_n) : \mathrm{Msg}}\ \textsc{Fun-Msg}$$

$$\frac{\Gamma; R \vdash t : \mathrm{T} \qquad \Gamma(\mathsf{k}) = \mathrm{SK[T]}}{\Gamma; R \sqcup \{\mathsf{r}\} \vdash \mathsf{senc}(t, \mathsf{k}[\vec{j}], \mathsf{r}[\vec{i}]) : \mathrm{Low}}\ \textsc{Senc}$$

$$\frac{\Gamma; R_1 \vdash t_1 : \mathrm{Msg} \qquad \Gamma; R_2 \vdash t_2 : \mathrm{Msg}}{\Gamma; R_1 \sqcup R_2 \vdash t_1 = t_2 : \mathrm{Bool}}\ \textsc{Eq}$$

$$\frac{\Gamma; R_1 \vdash t_1 : \mathrm{High} \qquad \Gamma; R_2 \vdash t_2 : \mathrm{Low}}{\Gamma; R_1 \sqcup R_2 \vdash t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}}\ \textsc{Eq-False}$$

$$\frac{\Gamma; R_1 \vdash t_1 : \mathrm{Cst}^0_{\mathsf{c}} \qquad \Gamma; R_2 \vdash t_2 : \mathrm{Cst}^0_{\mathsf{c}}}{\Gamma; R_1 \sqcup R_2 \vdash t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{true}}}\ \textsc{Eq-True-Cst}$$

$$\frac{\Gamma; R_i \vdash t_i : \mathrm{Cst}^\star_{\mathsf{c}_i} \qquad \text{with } i = 1, 2 \text{ and } \mathsf{c}_1 \neq \mathsf{c}_2}{\Gamma; R_1 \sqcup R_2 \vdash t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}}\ \textsc{Eq-False-Cst}$$

$$\frac{\Gamma; R_0 \vdash t : \mathrm{Bool} \qquad \Gamma; R_i \vdash t_i : \mathrm{T} \text{ with } i = 1, 2}{\Gamma; R_0 \sqcup R_1 \sqcup R_2 \vdash \mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathrm{T}}\ \textsc{If}$$

Fig. 1: Typing rules for macro-free meta-terms - Part I

$$\frac{\Gamma(x) = \mathrm{T}}{\Gamma; R \vdash x : \mathrm{T}}\ \textsc{Var} \qquad \frac{\Gamma; R \vdash t : \mathrm{Msg} \quad \Gamma(\mathsf{k}) = \mathrm{SK[T]}}{\Gamma; R \vdash \mathsf{sdec}(t, \mathsf{k}[\vec{j}]) : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}}\ \textsc{Sdec}$$

$$\frac{\Gamma; R \vdash t : \mathrm{T}_1 \times \mathrm{T}_2}{\Gamma; R \vdash \mathsf{fst}(t) : \mathrm{T}_1}\ \textsc{Fst} \qquad \frac{\Gamma; R \vdash t : \mathrm{T}_1 \times \mathrm{T}_2}{\Gamma; R \vdash \mathsf{snd}(t) : \mathrm{T}_2}\ \textsc{Snd}$$

$$\frac{\Gamma, x : \mathrm{T}'; R_1 \vdash t : \mathrm{T} \qquad \Gamma; R_2 \vdash t' : \mathrm{T}'}{\Gamma; R_1 \sqcup R_2 \vdash t[x \mapsto t'] : \mathrm{T}}\ \textsc{Assign}$$

$$\frac{\Gamma, x : \mathrm{T}_1; R_1 \vdash t : \mathrm{T} \qquad \Gamma, x : \mathrm{T}_2; R_2 \vdash t : \mathrm{T}}{\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; R_1 \sqcup R_2 \vdash t : \mathrm{T}}\ \textsc{Break-Sum}$$

$$\frac{\Gamma; R_0 \vdash t_0 : \mathrm{Cst}^0_{\mathsf{true}} \qquad \Gamma; R_1 \vdash t_1 : \mathrm{T}_1}{\Gamma; R_0 \sqcup R_1 \vdash \mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathrm{T}_1}\ \textsc{If-True}$$

$$\frac{\Gamma; R_0 \vdash t_0 : \mathrm{Cst}^0_{\mathsf{false}} \qquad \Gamma; R_2 \vdash t_2 : \mathrm{T}_2}{\Gamma; R_0 \sqcup R_2 \vdash \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathrm{T}_2}\ \textsc{If-False}$$

Fig. 2: Typing rules for macro-free meta-terms - Part II

$$\frac{}{\Gamma; R \vdash \mathsf{input@}T : \mathrm{Low}} \qquad \frac{}{\Gamma; R \vdash \mathsf{frame@}T : \mathrm{Low}}$$

$$\frac{}{\Gamma; R \vdash \mathsf{output@}T : \mathrm{Low}}$$

$$\frac{}{\Gamma; R \vdash \mathsf{cond@}T : \mathrm{Bool}} \qquad \frac{}{\Gamma; R \vdash \mathsf{exec@}T : \mathrm{Bool}}$$

Fig. 3: Typing rules for macros: IN, FRAME, OUT, COND, EXEC.

ZEROS deems public the string of zeros of the same length as any message. Rule SENC is a specific rule to handle the senc function symbol, which allows to type the resulting ciphertext Low even if the plaintext is not public. That rule enforces that *i)* the plaintext has the type prescribed by the key's type, and *ii)* the randomness of the encryption is only used once (it is removed from the typing environment). To enforce this last point, we also need to split the set of randoms in the environment when a rule creates several branches in the typing derivation, so that a given random is used only in one branch (see *e.g.* rule PAIR). Several rules type equality tests, *i.e.* $t_1 = t_2$. Rule EQ is the most generic, and simply gives type Bool to an equality test. Rules EQ-FALSE-... and EQ-TRUE-... are more precise. When the types of t_1 and t_2 indicate that they represent two different constant symbols, or that one is secret and the other public, we statically know that the condition evaluates to false. Note that we cannot conclude when t_1 and t_2 are indexed constants with the same symbol: they may be different or not, depending on the value of the indices. Lastly, rule IF allows to type a conditional when both branches have the same type.

This first set of rules produces rather simple typing derivations. The rules in Figure 2 allow more complex reasoning. They typically rely on information gained by the first set of rules, such as the type of equality tests or of ciphertexts, to eliminate conditional branches or apply destructors. They also allow reasoning on variables.

Rule VAR gives to a variable the type indicated in the environment. Rule ASSIGN stores a term in a new variable, and adds it to Γ. If term t contains a variable x, we let $t[x \mapsto t']$ denote the term obtained by substituting each occurrence x with t' in t. That substituted term may be typed by first typing t' with some type T, and then typing t in an environment where x is given type T. This is helpful to structure a typing proof, as well as to isolate a variable on which we will later perform a case disjunction using rule BREAK-SUM. Rules FST and SND allow to destruct a product type. Rule SDEC allows to infer the type of the decryption of any message of type Msg, relying on the type of the decryption key. Lastly, rules IF-TRUE and IF-FALSE allow to type a conditional in the specific case where we know that the test will be true (resp. false).

Example 6. *Continuing Example 5, consider the meta-term* $t \mathrel{\widehat{=}}$ $\mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ o_{\mathsf{S}[i,j,k]}\ \mathsf{else}\ \mathsf{empty}$ *representing the message output by the server's action. We have:*

$$\dfrac{\dfrac{\Pi_1 \qquad \Pi_2}{\Gamma_0', x : \mathrm{Msg} \times \mathrm{High} + \mathrm{Cst}^0_{\mathsf{fail}}; \{\mathsf{r_s}\} \vdash \mathsf{if}\ \phi^0_{\mathsf{S}}\ \mathsf{then}\ o^0_{\mathsf{S}}\ \mathsf{else\ empty} : \mathrm{Low}} \qquad \dfrac{\ldots}{\Gamma_0'; \emptyset \vdash t^{\mathsf{S}}_{\mathsf{sdec}} : \mathrm{Msg} \times \mathrm{High} + \mathrm{Cst}^0_{\mathsf{fail}}}\ \textsc{Sdec}}{\Gamma_0'; \{\mathsf{r_s}\} \vdash \mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ o_{\mathsf{S}[i,j,k]}\ \mathsf{else\ empty} : \mathrm{Low}}\ \textsc{Assign}$$

where ϕ^0_{S} and o^0_{S} are respectively $\phi_{\mathsf{S}[i,j,k]}$ and $o_{\mathsf{S}[i,j,k]}$ in which $t^{\mathsf{S}}_{\mathsf{sdec}}$ has been replaced with x.

Fig. 4: Typing derivation for Example 6.

$$\dfrac{\dfrac{\Pi_{\mathsf{cond}}}{\Gamma_0; \emptyset \vdash \phi_{\mathsf{S}[i,j,k]} : \mathrm{Bool}} \qquad \dfrac{\Pi_{\mathsf{then}}}{\Gamma_0; \{\mathsf{r_s}\} \vdash o_{\mathsf{S}[i,j,k]} : \mathrm{Low}} \qquad \dfrac{\Pi_{\mathsf{else}}}{\Gamma_0; \emptyset \vdash \mathsf{empty} : \mathrm{Low}}}{\Gamma_0; \{r_s\} \vdash \mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ o_{\mathsf{S}[i,j,k]}\ \mathsf{else\ empty} : \mathrm{Low}}$$

This typing derivation ends with an instance of rule IF. *The proof* Π_{then} *ends with an application of rule* SENC, *whereas* Π_{else} *is a simple application of rules* CST-0 *and* SUB-TYPING. *Recall that the operator* $\wedge$ *is encoded using* $\mathsf{if}\ \cdot\ \mathsf{then}\ \cdot\ \mathsf{else}\ \cdot$, *and thus* $\phi_{\mathsf{S}[i,j,k]}$ *can be typed using rules* IF *and* EQ.

To illustrate another aspect of the type system, consider now a different typing environment

$$\Gamma_0' = \{\mathsf{k}, \mathsf{k_{fresh}} : \mathrm{High},\ \mathsf{k_s} : \mathrm{SK}[\mathrm{Msg} \times \mathrm{High}]\}.$$

The meta-term t *can also be given type* Low *in* $(\Gamma_0'; R_0)$. *However, the previous typing derivation above no longer works. Indeed,* $\mathsf{k_s}$ *now has a more restrictive type, and typechecking the message* $o_{\mathsf{S}[i,j,k]}$ *needs some more careful work. Recall from Example 2 that, noting* $t^{\mathsf{S}}_{\mathsf{sdec}} \mathrel{\widehat{=}} \mathsf{sdec}(\mathsf{snd}(\mathsf{input@S}[i,j,k]), \mathsf{k_s}[i])$, *we have* $o_{\mathsf{S}[i,j,k]} = \mathsf{senc}(\langle \mathsf{a}[i], \mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})\rangle, \mathsf{k_s}[j], \mathsf{r_s}[i,j,k])$.

To obtain $\Gamma_0'; \{\mathsf{r_s}\} \vdash o_{\mathsf{S}[i,j,k]} : \mathrm{Low}$, *we must show that* $\langle \mathsf{a}[i], \mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})\rangle$ *has type* $\mathrm{Msg} \times \mathrm{High}$, *and thus that* $\mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})$ *has type* High. *Rule* SDEC *gives* $t^{\mathsf{S}}_{\mathsf{sdec}}$ *type* $\mathrm{T}_0 = \mathrm{Msg} \times \mathrm{High} + \mathrm{Cst}^0_{\mathsf{fail}}$, *and we thus perform a case disjunction, using rules* ASSIGN *and* BREAK-SUM *(on the top-left), as displayed in Figure 4. The typing derivation* Π_2 *deals with the case of type* $\mathrm{Cst}^0_{\mathsf{fail}}$: *in that case, rule* EQ-FALSE *ensures that the condition is false, and thus the resulting meta-term has type* Low *by rule* IF-FALSE. *The typing derivation* Π_1 *handles the case of type* $\mathrm{Msg} \times \mathrm{High}$. *As that is exactly the type required for key* $\mathsf{k_s}$ *in* Γ_0', *we easily conclude by rule* SENC.

C. Secrecy by typing

Our main result states that a well-typed protocol preserves secrecy of any term that can be typed High. More precisely, for any trace model $\mathbb{T}$, the translation in $\mathbb{T}$ of a meta-term t_H typed High cannot be deduced by the attacker from the translation of meta-terms typed Low (except with negligible probability). Before formally stating this result, we introduce the notions of *well-typed protocol* and *non-deducibility*.

Definition 3. *Let* $(\Gamma; R)$ *be a well-formed typing environment that does not contain variables, and* $\mathcal{P}$ *a protocol. We say that* $\mathcal{P}$ *is* well-typed *in* $(\Gamma; R)$ *when for each action*

$$\mathsf{A}[\vec{i}].(\phi_{\mathsf{A}[\vec{i}]}, o_{\mathsf{A}[\vec{i}]})$$

composing the protocol:

- $\Gamma; \emptyset \vdash \phi_{\mathsf{A}[\vec{i}]} : \mathrm{Bool}$*; and*
- $\Gamma; R_{\mathsf{A}} \vdash \mathsf{if}\ \phi_{\mathsf{A}[\vec{i}]}\ \mathsf{then}\ o_{\mathsf{A}[\vec{i}]}\ \mathsf{else\ empty} : \mathrm{Low}$*;*

for some sets $\{R_{\mathsf{A}}\}_{\mathsf{A} \in \mathcal{A}}$, *that form a partition of* R.

Example 7. *The protocol* $\mathcal{P}_{\mathsf{WMF}}$ *described in Example 1 is well-typed w.r.t. the environment* $(\Gamma_0'; R_0)$. *For instance, considering the action* $\mathsf{S}[i,j,k]$, *we have seen in Example 6 that* $\Gamma_0; \emptyset \vdash \phi_{\mathsf{S}[i,j,k]} : \mathrm{Bool}$, *and actually the same derivation can be done with* Γ_0' *instead of* Γ_0.

Regarding the meta-term $\mathsf{if}\ \phi_{\mathsf{S}[i,j,k,]}\ \mathsf{then}\ o_{\mathsf{S}[i,j,k]}\ \mathsf{else\ empty}$, *we have established in Example 6 that:*

$$\Gamma_0'; \{\mathsf{r_s}\} \vdash \mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ o_{\mathsf{S}[i,j,k]}\ \mathsf{else\ empty} : \mathrm{Low}.$$

A similar reasoning lets us conclude for all the other actions.

Definition 4. *Given two ground base-logic terms* t *and* t', t' *is* not deducible *from* t, *denoted* $t \blacktriangleright t'$, *if for any computational model* $\mathbb{M}$ *and PPTM* $\mathcal{A}$,

$$\mathbb{P}_\rho \left[\mathcal{A}(1^\eta, \rho_a, [\![t]\!]^{\mathbb{M}}(1^\eta, \rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta, \rho)\right] \in \mathsf{negl}(\eta).$$

Armed with these notions, we can now state the following soundness theorem.

Theorem 1. *Let* $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$ *be a well-formed typing environment,* $\mathcal{P}$ *a protocol well-typed in* $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$, t_L *and* t_H *be two meta-terms such that* $\Gamma_{\mathcal{P}}; \emptyset \vdash t_L : \mathrm{Low}$ *and* $\Gamma_{\mathcal{P}}; \emptyset \vdash t_H : \mathrm{High}$. *Let* $\mathbb{T}$ *be a trace model. Then* $(t_L)^{\mathbb{T}}_{\mathcal{P}} \blacktriangleright (t_H)^{\mathbb{T}}_{\mathcal{P}}$.

Note that t_L and t_H are terms from the meta-logic. Therefore, the theorem above allows us to express that a name n typed High using our type system cannot be deduced from $\mathsf{frame@}\tau$ (typed Low by our type system), and thus to express that n is not deducible from the knowledge obtained by the attacker along an execution of the protocol. In other words, as a direct consequence of Theorem 1, we have the following corollary.

Corollary 1. *Let* $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$ *be a well-formed environment,* $\mathcal{P}$ *a protocol well-typed in* $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$, *and* t_H *a ground meta-term such that* $\Gamma_{\mathcal{P}}; \emptyset \vdash t_H : \mathrm{High}$. *Then* t_H *is secret in* $\mathcal{P}$.

Example 8. *We can rely on Corollary 1 to analyse key secrecy for protocol* $\mathcal{P}_{\mathsf{WMF}}$. *As usual, we may want to consider the secrecy of the key* K *from the point of view of the initiator (resp. the server, or the responder). This can be done by applying the previous Corollary to one of the following secret terms, depending on the point of view we want to consider. We give below the terms* t_H *when considering the view point of the initiator and the server.*

- *Initiator:* $\mathsf{k}[i,j,k]$*;*
- *Server:* $\mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ \mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})\ \mathsf{else}\ \mathsf{k_{fresh}}$ *where* $\phi_{\mathsf{S}[i,j,k]}$ *and* $t^{\mathsf{S}}_{\mathsf{sdec}}$ *are given in Example 2;*

$$\dfrac{\dfrac{\Pi_1}{\Gamma_0', x : \mathrm{T}_0; \emptyset \vdash t_0 : \mathrm{High}} \quad \dfrac{\Pi_2}{\Gamma_0', x : \mathrm{Cst}^0_{\mathsf{fail}}; \emptyset \vdash t_0 : \mathrm{High}}\ \textsc{Break-Sum}}{\Gamma_0', x : \mathrm{T}_0 + \mathrm{Cst}^0_{\mathsf{fail}}; \emptyset \vdash t_0 : \mathrm{High}} \quad \dfrac{\dfrac{\dfrac{\Gamma_0'; \emptyset \vdash \mathsf{input@S}[i,j,k] : \mathrm{Low}}{\Gamma_0'; \emptyset \vdash \mathsf{input@S}[i,j,k] : \mathrm{Msg}}\ \textsc{Sub-Typing}}{\Gamma_0'; \emptyset \vdash \mathsf{snd}(\mathsf{input@S}[i,j,k]) : \mathrm{Low}}\ \textsc{Fun-Msg} \quad \Gamma_0'(\mathsf{k_s}) = \mathrm{SK}[\mathrm{T}_0]}{\Gamma_0'; \emptyset \vdash t^{\mathsf{S}}_{\mathsf{sdec}} : \mathrm{T}_0 + \mathrm{Cst}^0_{\mathsf{fail}}}\ \textsc{Sdec}$$
$$\Gamma_0', \emptyset \vdash \mathsf{if}\ \phi_{\mathsf{S}[i,j,k]}\ \mathsf{then}\ \mathsf{snd}(t^{\mathsf{S}}_{\mathsf{sdec}})\ \mathsf{else}\ \mathsf{k}_{\mathsf{fresh}} : \mathrm{High}\ \textsc{Assign}$$

where $\mathrm{T}_0 = \mathrm{Msg} \times \mathrm{High}$, and $t_0 \mathrel{\widehat{=}} \mathsf{if}\ \phi^0_{\mathsf{S}}\ \mathsf{then}\ \mathsf{snd}(x)\ \mathsf{else}\ \mathsf{k}_{\mathsf{fresh}}$ with ϕ^0_{S} the term $\phi_{\mathsf{S}[i,j,k]}$ in which occurrences of $t^{\mathsf{S}}_{\mathsf{sdec}}$ are replaced by x.

The typing derivation for Π_2 is as follows where $\Gamma_{\mathsf{fail}} = \Gamma_0', x : \mathrm{Cst}^0_{\mathsf{fail}}$, and Π_1 can be derived in a similar way.

$$\dfrac{\dfrac{\Pi_3}{\Gamma_{\mathsf{fail}}; \emptyset \vdash \phi^0_{\mathsf{S}} : \mathrm{Cst}^0_{\mathsf{false}}}\ \textsc{Eq-False-Cst} \quad \dfrac{\Gamma_{\mathsf{fail}}(\mathsf{k}_{\mathsf{fresh}}) = \mathrm{High}}{\Gamma_{\mathsf{fail}}; \emptyset \vdash \mathsf{k}_{\mathsf{fresh}} : \mathrm{High}}\ \textsc{Name}}{\Gamma_0', x : \mathrm{Cst}^0_{\mathsf{fail}}; \emptyset \vdash t_0 : \mathrm{High}}\ \textsc{If-False}$$

Fig. 5: Typing tree for secrecy in WMF (Server view point)

Although these terms may seem rather complex, they are fairly intuitive to write in the CCSA *logic, and are in fact what one would naturally write in* SQUIRREL *when modelling the protocol. The purpose of* $\mathsf{k}_{\mathsf{fresh}}$ *is to have a dummy term to use when decryption fails, for which secrecy trivially holds.*

In each case, we can show that $\Gamma_0'; \emptyset \vdash t_H : \mathrm{High}$. *Therefore, by Corollary 1,* t_H *is secret in* $\mathcal{P}_{\mathsf{WMF}}$, i.e. *for any trace model* $\mathbb{T}$ *and computational model* $\mathbb{M}$, *no PPTM attacker* $\mathcal{A}$ *can compute with non-negligible probability the value of* t_H *from the messages he has seen during the execution of* $\mathcal{P}$ *(that is,* $\mathsf{frame@}\tau$, *for an arbitrary* τ*).*

Considering the view point of the initiator, i.e. $t_H = \mathsf{k}[i,j,k]$, *it is easy to see that* $\Gamma_0'; \emptyset \vdash t_H : \mathrm{High}$ *using the rule* NAME. *The derivation corresponding to the view point of the server is more complex, and is detailed in Figure 5. This derivation is another illustration of the use of the rules* ASSIGN *and* BREAK-SUM. *Note that the derivation* Π_3 *is actually performed using rules* EQ-TRUE *and* IF-TRUE, *as the conditional* ϕ^0_{S} *is of the form* $\mathsf{if}\ x = \mathsf{fail}\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \ldots$ *(once boolean connectors are inlined) and* $x : \mathrm{Cst}^0_{\mathsf{fail}}$ *in the current typing environment* Γ_{fail}.

V. PROOF OF SOUNDNESS OF THE TYPE SYSTEM

We prove the soundness of our type system, *i.e.* Theorem 1, in three main steps, in line with the way we introduced the typing rules in Section IV. These steps are summarised below, and the fully detailed proofs can be found in Appendix. Though we explain the proof in a simplified setting here (without states), the actual proofs are done considering the full meta-logic featuring states. All the definitions and theorems stated in this section are actually valid in both settings.

1) We first establish the soundness of a fragment which we call the *restricted type system*, composed of the rules given in Figure 1, specialised for base-logic terms, *i.e.* terms without macros or indices. Notably, the restricted system does not contain rules for decryption or destructors, making it easier to write cryptographic reductions.
2) We then leverage this soundness result, still considering base terms, to additionally deal with typing rules given in Figure 2. This involves rewriting base-logic terms to remove destructors, so that they can be handled by the restricted system, while preserving their semantics.
3) Lastly, we lift the soundness result from the base logic to the full meta-logic type system given in Figure 3, which mainly consists in considering macros, as well as meta-terms with indices.

A. Soundness of the restricted type system

The *restricted type system*, whose associated type judgement is denoted by $\vdash_r$, consists of the typing rules from Figure 1, with two major differences. First, only base-logic terms are considered, *i.e.* all indices are removed from the rules. Second, instead of a typing environment, the restricted type system uses a *mapping environment*, denoted $(\Gamma; \mathcal{R})$. Rather than a set of random symbols, $\mathcal{R}$ is a mapping that indicates explicitly how each random symbol must be used: if $\mathcal{R}(r) = (m, k)$, then the type system only accepts the use of r inside an encryption of the form $\mathsf{senc}(m, k, r)$. Carrying this additional information makes cryptographic proofs easier, by ensuring consistent use of encryption randomness. It also alleviates the need to split the random sets to ensure their unique use. Note that the mapping environment $\mathcal{R}$ is only an intermediate step of the proof, but does not need to be user-provided in the end.

The soundness theorem for the restricted system, informally, states that if a term is given type T, then it is also semantically of type T, in the sense that its computational interpretation (in any model) satisfies semantic conditions expressing the intended meaning of type T. These conditions, given below, use the following notion of *semantic equivalence*: for a ground base term t and a finite set of such terms S, we write $t \mathrel{\tilde{\in}} S$ if for all model $\mathbb{M}$, $\mathbb{P}_\rho \left[\exists t' \in S. \llbracket t \rrbracket^{\mathbb{M}}(1^\eta, \rho) = \llbracket t' \rrbracket^{\mathbb{M}}(1^\eta, \rho)\right] \in \mathsf{ow}(\eta)$. We write $t \approx t'$ when $t \mathrel{\tilde{\in}} \{t'\}$.

Definition 5. *Given a well-formed* $(\Gamma; \mathcal{R})$, *a ground base-logic term* t, *and a message type* T, *we say that* t is in the interpretation of T w.r.t. $(\Gamma; \mathcal{R})$, *denoted by* $\Gamma; \mathcal{R} \models t : \mathrm{T}$, *when there exists* $t' \approx t$ *such that* $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}$, *and:*

- *if* $\mathrm{T} = \mathrm{Cst}^0_{\mathsf{c}}$: $t \approx \mathsf{c}$;
- *if* $\mathrm{T} = \mathrm{Cst}^\infty_{\mathsf{c}}$: $t \mathrel{\tilde{\in}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$;

- *if* $\mathrm{T} = \mathrm{Bool}$*:* $t \mathrel{\tilde{\in}} \{\mathsf{true}, \mathsf{false}\}$*;*
- *if* $\mathrm{T} = \mathrm{Msg}$ *or* $\mathrm{T} = \mathrm{Low}$*: no further condition;*
- *if* $\mathrm{T} = \mathrm{High}$*: for all* t_L*, if* $\Gamma; \mathcal{R} \models t_L : \mathrm{Low}$ *then* $t_L \blacktriangleright t$*;*
- *if* $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$*:* $\Gamma; \mathcal{R} \models \mathsf{fst}(t) : \mathrm{T}_1$*, and* $\Gamma; \mathcal{R} \models \mathsf{snd}(t) : \mathrm{T}_2$*;*
- *if* $\mathrm{T} = \mathrm{T}_1 + \mathrm{T}_2$*: either* $\Gamma; \mathcal{R} \models t : \mathrm{T}_1$*, or* $\Gamma; \mathcal{R} \models t : \mathrm{T}_2$*, or there exist* b*,* t_1*,* t_2 *such that* $t' = \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$*,* $\Gamma; \mathcal{R} \models b : \mathrm{Bool}$*, and* $\Gamma; \mathcal{R} \models t_i : \mathrm{T}_i$ *for* $i = 1, 2$*.*

This notion effectively defines for each type T its interpretation: a set of terms that are not only syntactically of type T (up to $\approx$), but also semantically behave as expected of type T, in any model $\mathbb{M}$. The interpretations of $\mathrm{Cst}^\star_\mathsf{c}$, Bool, pair types, and Msg are rather unsurprising. For instance, Bool's contains all terms that always evaluate to either $[\![\mathsf{true}]\!]^{\mathbb{M}}$ or $[\![\mathsf{false}]\!]^{\mathbb{M}}$. The interpretation of sum type $\mathrm{T}_1 + \mathrm{T}_2$ is more subtle: it contains those of T_1 and T_2, as well as terms equivalent to a conditional, with one branch in T_1's interpretation and the other in T_2's. The crucial point is the interpretation of types Low and High. We do not impose semantic conditions for Low: any term of type Low is in its interpretation. In particular, we do not require that term to be effectively deducible by an attacker. The interpretation of type High, however, only contains messages that *cannot* be deduced from *any* Low term. With this definition, it is clear that all terms in the interpretation of High are actual secrets: they cannot be computed using publicly available information.

We establish that the restricted type system is sound: a term is of type T only if it is semantically in T's interpretation.

Theorem 2. *Let* $(\Gamma; \mathcal{R})$ *be a well-formed mapping environment,* t *a ground base-logic term, and* T *a message type.*
If $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$*, then* $\Gamma; \mathcal{R} \models t : \mathrm{T}$*.*

The central point is to show why an attacker, knowing a public term t_L, of type Low, cannot find the value of a secret nonce n, of type High. We first establish that the restricted system guarantees that n can only appear in t_L in certain contexts: in equality tests, under the zeros symbol, or in ciphertexts encrypted with secret keys.

We then construct a term t'_L, by replacing each ciphertext built with a secret key $\mathsf{senc}(m, k, r)$ occurring in t_L with $\mathsf{senc}(\mathsf{zeros}(m), k, r)$: the encryption of a string of $|m|$ zeros.

We show that, if n was deducible from t_L, then it would also be from t'_L, by reducing the IND-CPA assumption on the encryption scheme. An important point of that reduction is to show that the interpretation of t_L can be simulated by the adversary against IND-CPA. We do so by constructing an *evaluator*, *i.e.* a PPTM that can compute the interpretation of any term that is typable in the restricted system, while having access to the oracles provided by the IND-CPA game, and all names except secret keys and encryption randomness. Note that this is only possible thanks to the restricted system forbidding decryption operations.

We then conclude by showing that n is in fact *not* deducible from t'_L. Indeed, it only appears in that term under the zeros symbols or in equality tests, that both only leak a number of bits of information insufficient to obtain all η bits of n.

B. Soundness of the base-level type system

The base-level type system is composed of all the rules in Figures 1 and 2, adapted to base-logic terms by removing all indices. The associated type judgement is $\vdash_b$. Unlike the restricted system, it uses typing environments $(\Gamma; R)$ where R is a set of random symbols, rather than mapping environments.

We establish the following soundness theorem for the base-level system.

Theorem 3. *Let* $(\Gamma; R)$ *be a well-formed typing environment,* t *a ground term of the base logic, and* T *a message type. Then* $\Gamma; R \vdash_b t : \mathrm{T}$ *implies* $\Gamma; \mathcal{R} \models t : \mathrm{T}$ *for some* $\mathcal{R}$*.*

The main idea is to prove the existence of a term $t' \approx t$, with (overwhelmingly) the same semantics as t, which is typable in the restricted system. In other words, a term such that $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}$ for some mapping $\mathcal{R}$. We can then apply Theorem 2 to conclude that $\Gamma; \mathcal{R} \models t' : \mathrm{T}$, and therefore $\Gamma; \mathcal{R} \models t : \mathrm{T}$, as t and t' have the same interpretation.

The difficulty, when building the term t', is that t may contain constructions that are only typable using the added rules in Figure 2, that are not present in the restricting system. In particular, t may contain decryption operations, that must somehow be discarded to make t' typable. To solve this issue, we use the INT-CTXT assumption, which basically tells us that the only terms that can be successfully decrypted with the secret key k are those encrypted with k earlier. Formally, if $\mathsf{senc}(p_1, k, r_1), \ldots, \mathsf{senc}(p_\ell, k, r_\ell)$ denote the encryptions with k occurring as subterms of t, we show that:

$$\mathsf{sdec}(t, k) \approx \begin{cases} \mathsf{if}\ t = \mathsf{senc}(p_1, k, r_1)\ \mathsf{then}\ p_1\ \mathsf{else} \\ \quad \ldots \\ \quad \mathsf{if}\ t = \mathsf{senc}(p_\ell, k, r_\ell)\ \mathsf{then}\ p_\ell\ \mathsf{else}\ \mathsf{fail}. \end{cases}$$

That (overwhelming) equality is proved by reduction of INT-CTXT. Similarly to the previous Theorem, we construct PPTMs that compute all terms involved, and use them to construct an adversary that would win the INT-CTXT game if the equation did not hold.

Replacing all forbidden subterms in similar ways, we construct the term t' and conclude the proof.

C. Soundness of the entire type system

Finally, it remains to lift the soundness of the base type system to the entire meta-level type system. Roughly, we show that for any protocol $\mathcal{P}$, any trace model $\mathbb{T}$, and any meta-term t with type T in the entire system, the base-logic term $(t)^{\mathbb{T}}_{\mathcal{P}}$ has type T in the base-level type system. The soundness result, Theorem 1, then follows.

That proof does not rely on cryptographic arguments. Basically, $(t)^{\mathbb{T}}_{\mathcal{P}}$ is defined as $\bar{t}^{\mathbb{T}}\theta^{\mathbb{T}}_{\mathcal{P}}$, with $\bar{t}^{\mathbb{T}}$ being t where macros are replaced with dedicated variables, and $\theta^{\mathbb{T}}_{\mathcal{P}}$ mapping these to the macros' bodies (recursively expanding macros they contain – see Section A-B2). Accordingly, we typecheck $(t)^{\mathbb{T}}_{\mathcal{P}}$ by first using rule ASSIGN to introduce these variables, and typecheck separately $\bar{t}^{\mathbb{T}}$ (which no longer contains macros) and each macro's body (recursively replacing again the macros it contains with variables). The trace model $\mathbb{T}$ provides an

order in which the macros can be replaced so as to ensure disjointness of the sets of random symbols used in the typing derivation at each step.

VI. Case studies

We first give some details about our implementation and the heuristics we use for performing the type-checking. Then, we review several protocols of the literature and of the ISO/IEC 11770 standard [21]. We discuss the corruption scenario and the secrecy properties that are considered before summarising our findings.

A. Implementation

We implement our approach within SQUIRREL through the development of a new tactic called `typing`. The resulting modified version of SQUIRREL is available in the supplementary material of this publication. Our additions represent 1600 lines of OCaml code.

When modeling a protocol in the SQUIRREL syntax, the user has to provide some light typing annotations when declaring names. It can be the annotation `Rand`, to indicate that the name will be used as a random in an encryption, or a type such as Low, High, $\mathrm{SK}[\mathrm{Cst}_{\mathsf{a}}^{\infty} \times \mathrm{High}]$. When a protocol is declared (using the keyword `system` in SQUIRREL), the implementation will check automatically that the protocol under study is well-typed w.r.t. the typing annotations provided by the user. If so, the user is later on allowed to apply the `typing` tactic. In a proof, `typing Eq` can be called on an hypothesis `Eq` of the form $t_H = t_L$. It performs all the checks required to give type High to t_H and Low to t_L. If it succeeds, thanks to Theorem 1, we know that `H` is contradictory, and thus the current goal holds and can be closed. Therefore, our procedure boils down to finding a typing derivation corresponding to a typing judgement. The choice of the typing rules to apply is guided by the syntax of the term we have to type, and aims at finding types that are as precise as possible. For instance, in order to type $\mathsf{f}(t)$, even though both Fun-Low and Fun-Msg could be used, we will use Fun-Low when possible, as it leads to a more precise type.

The only rules whose applications are not completely guided by the syntax are Assign and Break-Sum. Regarding the rule Assign, when typing a term, we start by applying the rule Assign to each of its subterms – doing so when manually typing is unpleasant, as it generates many intermediary variables, but it is convenient when writing our procedure. The rule Break-Sum must be used more sparingly. Indeed, recall that this rule partitions the set of randoms in two. Hence, when it is applied, we have to ensure that the same random will not be used on both branches. In some cases, this could make it impossible to type the branches. The strategy consisting of applying Break-Sum as soon as possible is thus not always the right way to obtain a typing derivation. Our heuristic consists of a first attempt where we apply rule Break-Sum as soon as possible, and, if that fails (*i.e.* randoms are not used in the right way), to make a second attempt, where we forbid Break-Sum applications that have led to the misuse of randoms. This is not a complete typing procedure: it may sometimes fail although a derivation exists. Nevertheless, our implementation is precise enough to type all our case studies. Actually, the "first attempt" of the heuristic above was already sufficient to analyse all symmetric protocols reported in this section.

In practice, SQUIRREL's syntax differs slightly from the presentation in this paper – for simplicity, we omitted some details that the implementation accounts for. For instance, SQUIRREL users can declare several encryption functions, while we assumed only one. This is not an issue, as our results still hold with the same proofs, provided any given key is always used with the same function. Hence, our implementation lets users specify in a key's type which encryption function it should be used with. As another example, while we interpret all function symbols as PPTMs, SQUIRREL allows for non-polynomial symbols in its recent extensions – our typechecker simply launchs a warning and assumes no such symbols are used.

B. Scenario and corruption model

The scenario we have presented so far for WMF is rather simple. It models an arbitrary number of agents $\mathsf{a}[i]$, but assumes they are all honest. This is too simplistic, and may lead to some attacks being missed. We therefore consider a more realistic scenario, involving corrupted agents. For WMF, we model an arbitrary number of honest agents $\mathsf{a}[i]$ each sharing a key $\mathsf{k_s}[i]$ with the server S, as well as an arbitrary number of dishonest agents $\mathsf{a^d}[j]$ each sharing a key $\mathsf{k_s^d}[j]$ with S. The system of actions modelling the scenario with corruption includes 6 extra actions – in addition to the 5 actions detailed in Example 2. We add an action to model an initiator willing to engage in communication with a dishonest agent $\mathsf{a^d}[j]$. In that case, the session key and random used are modelled as names of type Low: $\mathsf{k^d}[i,j,k]$, and $\mathsf{r_a^d}[i,j,k]$. Similarly, an extra action models the responder talking to a dishonest agent. The other four extra actions model the server. We have two actions for modelling the two branches of the conditional when the server is in communication with an initiator who is dishonest, and similarly we consider the case where the responder is dishonest. As usual in formal verification, roles that can be simulated by the attacker need not be explicitly modelled.

With this scenario in mind, we can show that the resulting protocol is well-typed w.r.t. the environment $(\Gamma_{\mathsf{WMF}}; R_{\mathsf{WMF}})$ where $R_{\mathsf{WMF}} = \{\mathsf{r_a}, \mathsf{r_s}, \mathsf{r_a^d}, \mathsf{r_s^{dh}}\}$ and Γ_{WMF} is as follows:

$$\mathsf{k_s} : \mathrm{SK}[\mathrm{Cst}_{\mathsf{a}}^{\infty} \times \mathrm{High} + \mathrm{Cst}_{\mathsf{a^d}}^{\infty} \times \mathrm{Low}],$$
$$\mathsf{k}, \mathsf{k_{fresh}} : \mathrm{High}, \quad \mathsf{k_s^d}, \mathsf{k^d}, \mathsf{r_s^{hd}} : \mathrm{Low}$$

The keys $\mathsf{k_s^d}[i]$, $\mathsf{k^d}[i,j,k]$, as well as the randoms $\mathsf{r_s^{hd}}[i,j,k]$ are modelled using names having type Low. They are indeed meant for use in sessions with corrupted agents, and are thus safe to emit.

C. Secrecy properties

The security property under study is the secrecy (non-deducibility) of the session key. As usual, this secrecy property depends on which role's point of view we consider. In our

case study, we consider secrecy from the point of view of each role (initiator, server, responder). We first illustrate the three secrecy properties on our running example:

- Initiator: $t_H^{\mathsf{I}} \stackrel{\mathsf{def}}{=} \mathsf{k}[i,j,k]$;
- Server: $t_H^{\mathsf{S}} \stackrel{\mathsf{def}}{=} \mathsf{if}\ \phi_{\mathsf{S}}\ \mathsf{then}\ \mathsf{snd}(t_{\mathsf{sdec}}^{\mathsf{S}})\ \mathsf{else}\ \mathsf{k}_{\mathsf{fresh}}$;
- Responder: $t_H^{\mathsf{R}} \stackrel{\mathsf{def}}{=} \mathsf{if}\ \phi_{\mathsf{R}}\ \mathsf{then}\ \mathsf{snd}(t_{\mathsf{sdec}}^{\mathsf{R}})\ \mathsf{else}\ \mathsf{k}_{\mathsf{fresh}}$.

where $\phi_{\mathsf{R}} = \neg(t_{\mathsf{sdec}}^{\mathsf{R}} = \mathsf{fail}) \wedge \mathsf{fst}(t_{\mathsf{sdec}}^{\mathsf{R}}) = \mathsf{a}[i]$ with $t_{\mathsf{sdec}}^{\mathsf{R}} = \mathsf{sdec}(\mathsf{input@R}[i,j,k], \mathsf{k}_{\mathsf{s}}[j])$. The formula ϕ_{S} and the term $t_{\mathsf{sdec}}^{\mathsf{S}}$ are given in Examples 2.

Depending on which role's point of view is considered, the typing environment and the term t_H^{X} expressing secrecy may differ. In the rest of the section, we consider the three secrecy properties together, and we provide an environment in which the protocol, as well as t_H^{X} for $\mathsf{X} \in \{\mathsf{I},\mathsf{S},\mathsf{R}\}$, can be typed.

D. Review of symmetric key protocols

We study 6 symmetric key protocols described in [14], as well as some from the ISO standard 11770 pt. II [21]. They all aim at exchanging a session key K_{ab}. Most rely on a trusted server S, and long-term keys that each agent shares with S. These case studies have never been done in SQUIRREL before, and as in the case of WMF, establishing secrecy (without the `typing` tactic) will require dozens of lines of code.

For some of the protocols, as done *e.g.* in [15], we removed the last step, which consists in confirming the exchanged key by using it to encrypt a message. Our framework currently only covers encryption with long-term (fixed) keys. These cases are marked with $\star$. We also sometimes tweaked the protocols, to include explicit tags. This consists in adding a constant in some encrypted messages, to avoid confusion – and is good practice in protocol design. Note that, for protocols of the 11770 standard, tags are already present, and key confirmation steps are indicated as optional, which legitimises us not to take them into account. The standard defines 13 protocols based on symmetric encryption. These are divided into three categories (point to point key establishment, mechanisms that use a key distribution centre, and mechanisms that use a key translation centre). Protocols within each category are similar, and we thus chose to only analyse one per category.

Our findings are summarised in Table I. Even if the analysis is performed automatically by our tool once the type annotations are provided by the user, we detail below how our typing system works on the Wide Mouthed Frog protocol, and give some details in Appendix F regarding the others. As explained in Section II, the proof scripts are rather short (a few tactics are sufficient to conclude). We do not indicate any timing information but the analysis is done in less than one second.

E. Wide Mouthed Frog protocol

The well-formed environment $(\Gamma_{\mathsf{WMF}}; R_{\mathsf{WMF}})$ described in Section VI-B lets us typecheck the protocol, as well as the terms t_H^{X}, *i.e.* $\Gamma_{\mathsf{WMF}}; R_{\mathsf{WMF}} \vdash t_H^{\mathsf{X}} : \mathrm{High}$ for $\mathsf{X} \in \{\mathsf{I},\mathsf{S},\mathsf{R}\}$. Note that, to type ciphertexts using dishonest keys, we need rule FUN-LOW, and thus we ensure that the plaintext has type Low. For instance, for action $\mathsf{S}^{\mathsf{hd}}[i,j,k]$ modelling the server playing with $\mathsf{a}[i]$ and $\mathsf{a}^{\mathsf{d}}[j]$, we must give type Low to the term:

$$\mathsf{if}\ \neg(t_{\mathsf{sdec}}^{\mathsf{hd}} = \mathsf{fail}) \wedge (\mathsf{fst}(t_{\mathsf{sdec}}^{\mathsf{hd}}) = \mathsf{a}^{\mathsf{d}}[j])\ \mathsf{then}\ \mathsf{senc}(\langle \mathsf{a}[i], \mathsf{snd}(t_{\mathsf{sdec}}^{\mathsf{hd}})\rangle, \mathsf{k}_{\mathsf{s}}^{\mathsf{d}}[j], \mathsf{r}_{\mathsf{s}}^{\mathsf{hd}}[i,j,k])\ \mathsf{else}\ \mathsf{empty}$$

where $t_{\mathsf{sdec}}^{\mathsf{hd}} \mathrel{\widehat{=}} \mathsf{sdec}(\mathsf{snd}(\mathsf{input@S}^{\mathsf{hd}}[i,j,k]), \mathsf{k}_{\mathsf{s}}[i])$. The plaintext contains $\mathsf{a}[i]$, a constant that can be typed Low by SUB-TYPING. It remains to show that $\mathsf{snd}(t_{\mathsf{sdec}}^{\mathsf{hd}})$ has type Low as well. With SDEC (and FUN-MSG), we can give $t_{\mathsf{sdec}}^{\mathsf{hd}}$ the type:

$$(\mathrm{Cst}_{\mathsf{a}}^{\infty} \times \mathrm{High} + \mathrm{Cst}_{\mathsf{a}^{\mathsf{d}}}^{\infty} \times \mathrm{Low}) + \mathrm{Cst}_{\mathsf{fail}}^{0}.$$

The first and last options are discarded by exploiting the test $\mathsf{fst}(t_{\mathsf{sdec}}^{\mathsf{hd}}) = \mathsf{a}^{\mathsf{d}}[j]$ (resp. $\neg(t_{\mathsf{sdec}}^{\mathsf{hd}} = \mathsf{fail})$) of the server. As only the second one remains, $\mathsf{snd}(t_{\mathsf{sdec}}^{\mathsf{hd}})$ can be typed Low.

Thanks to Theorem 1, we conclude that secrecy of the key holds (from the points of view of the initiator, server, and responder), in the sense that it cannot be computed by any PPTM attacker, except with negligible probability.

	without tag	with explicit tags
Wide Mouthed Frog	✓	✓
Denning Sacco	✗	✓
Otways-Rees	✗	✓
Needham-Schroeder⋆	✗	✓
Yahalom⋆	✗	✓
Yahalom-Paulson⋆	✗	✓
Mechanism 6	-	✓
Mechanism 9	-	✓
Mechanism 13	-	✓

✓ : Well-typed protocol ; ✗ : Typing impossible ; - : Not defined
Mechanisms 6, 9, 13 are from the ISO-11770 standard.

TABLE I: Summary of our results

VII. DEALING WITH ASYMMETRIC ENCRYPTION

In this section, we showcase the extensibility of our type system by adding support for IND-CCA-2[1] asymmetric encryption. We assume three new function symbols: $\mathsf{aenc}/3$ and $\mathsf{adec}/2$, interpreted respectively as the encryption and decryption algorithms of an asymmetric scheme, and $\mathsf{pk}/1$ that produces the public encryption key associated to a private key.

A. Modification of the type system

We add a new type for asymmetric private keys: AK[T]. As in the symmetric case, it indicates the type of plaintexts intended to be encrypted with the associated public key, which gives information when decrypting. Public encryption keys, on the other hand, are typed Low. We add three new typing rules:

$$\frac{\Gamma(\mathsf{k}) = \mathrm{AK[T]}}{\Gamma; R \vdash \mathsf{pk}(\mathsf{k}[\vec{j}]) : \mathrm{Low}}\ \textsc{Pk}$$

$$\frac{\Gamma; R \vdash t : \mathrm{T} \qquad \Gamma(\mathsf{k}) = \mathrm{AK[T]}}{\Gamma; R \sqcup \{\mathsf{r}\} \vdash \mathsf{aenc}(t, \mathsf{pk}(\mathsf{k}[\vec{j}]), \mathsf{r}[\vec{i}]) : \mathrm{Low}}\ \textsc{Aenc}$$

$$\frac{\Gamma; R \vdash t : \mathrm{Low} \qquad \Gamma(\mathsf{k}) = \mathrm{AK[T]}}{\Gamma; R \vdash \mathsf{adec}(t, \mathsf{k}[\vec{j}]) : \mathrm{T} + \mathrm{Low}}\ \textsc{Adec}$$

[1]The IND-CCA-2 game is given in Appendix.

Note that these rules are added to the type system but do not replace the earlier ones: all previous typing rules remain present. Except for the possibility to publish the encryption key, the main difference to the symmetric case is the decryption rule. As the encryption key is public, the attacker can use it to encrypt messages of his own. Decryption hence either returns an honest message of type T, or a public message of type Low (the decryption failure case is included in type Low).

In the restricted system, as before, we wish to forbid the use of decryption, as it complicates cryptographic reductions. More precisely, messages encrypted by the protocol are used as challenges in IND-CCA-2 reductions, and so must never be decrypted when doing the reduction. To do so, we define a new function symbol $\mathsf{adec}^\star(c, k, \langle t_i \rangle_{1 \le i \le \ell})$, taking as arguments a ciphertext, a private key, and a list of ciphertexts (encoded as nested pairs). Its semantics is as follows. If $[\![c]\!]^{\mathbb{M}}(1^\eta, \rho) = [\![t_i]\!]^{\mathbb{M}}(1^\eta, \rho)$ for some i, it returns $[\![\mathsf{fail}]\!]^{\mathbb{M}}$. Otherwise, it decrypts c with k. This way, if we ensure that all honest ciphertexts are in $\langle t_i \rangle_{1 \le i \le \ell}$, then $\mathsf{adec}^\star$ only decrypts attacker-produced ciphertexts, and returns a result of type Low. We use $\mathsf{adec}^\star$ to remove uses of adec in the restricted system.

The soundness proof requires some modifications to handle the additional rules. We give here a summary of the changes, and details can be found in Appendix G.

Restricted type system. The soundness proof is similar to the symmetric case. We replace the plaintexts m in the term known by the attacker with $\mathsf{zeros}(m)$. When applied to asymmetric encryption, this replacement is sound, as we can otherwise construct an adversary that wins the IND-CCA-2 game. That adversary needs to compute the interpretation of terms, and as in the symmetric case we show that there exists an evaluator that can do so for well-typed terms – using in particular the IND-CCA-2 decryption oracle to evaluate $\mathsf{adec}^\star$.

Base type system. We show, by correctness of the encryption scheme, the following transformation, which is then used to remove the symbol adec, as we did for senc in Section V-B.

$$\mathsf{adec}(t,k) \approx \begin{cases} \mathsf{if}\ t = \mathsf{aenc}(p_1, \mathsf{pk}(k), r_1)\ \mathsf{then}\ p_1\ \mathsf{else} \\ \quad \cdots \\ \quad \mathsf{if}\ t = \mathsf{aenc}(p_\ell, \mathsf{pk}(k), r_\ell)\ \mathsf{then}\ p_\ell\ \mathsf{else} \\ \quad\quad \mathsf{adec}^\star(t, k, \langle t_i \rangle_{1 \le i \le n}) \end{cases}$$

Meta-level type system. It remains to lift the soundness of the base system to the meta-logic type system: that proof remains the same as in the symmetric case.

B. Additional case studies

We apply our extended framework to two public-key protocols: mechanism 6 of the ISO-11770 standard (part III) [22], as well as the tagged version of the Needham-Schroeder-Lowe protocol (NSL) [25], informally described below.

$$\begin{aligned} A \to B: &\quad \{1, N_a, A\}_{\mathsf{pk}(B)} \\ B \to A: &\quad \{2, N_a, N_b, B\}_{\mathsf{pk}(A)} \\ A \to B: &\quad \{3, N_b\}_{\mathsf{pk}(B)} \end{aligned}$$

It is meant to ensure mutual authentication of A and B, through the secrecy of two nonces N_a and N_b. We consider here the case of N_b. To typecheck it, we give type $\mathrm{AK}[\mathrm{T}_0 + \mathrm{Low}]$ to the private key $\mathsf{ska}[i]$ of honest agent $\mathsf{a}[i]$, where T_0 is

$$\begin{aligned} & \mathrm{Cst}_1^0 \times \mathrm{High} \times \mathrm{Cst}_{\mathsf{a}}^\infty \\ + \quad & \mathrm{Cst}_2^0 \times \mathrm{Msg} \times \mathrm{High} \times \mathrm{Cst}_{\mathsf{a}}^\infty \\ + \quad & \mathrm{Cst}_3^0 \times \mathrm{High}. \end{aligned}$$

The model of NSL is made of two actions per role, with two cases for each: an agent executing the role can be talking with an honest agent or a dishonest one. Hence, it contains 8 actions in total, plus init. To illustrate the use of our type system, we detail below the typing of the initiator's second action, running a session with a dishonest responder. In that case, the term output by $\mathsf{a}[i]$ responding to $\mathsf{a}^\mathsf{d}[j]$ in session k is:

$$\mathsf{aenc}(\langle 3, \mathsf{thd}(t^\mathsf{d}_{\mathsf{adec}}) \rangle, \mathsf{pk}(\mathsf{skb}^\mathsf{d}[j]), \mathsf{r}^\mathsf{d}_\mathsf{a}[i,j,k])$$

where $t^\mathsf{d}_{\mathsf{adec}} = \mathsf{adec}(\mathsf{input@l}^\mathsf{d}_1[i,j,k], \mathsf{ska}[i])$, $\mathsf{thd}(\cdot)$ is a shortcut for $\mathsf{fst}(\mathsf{snd}(\mathsf{snd}(\cdot)))$ (retrieving the third element of a tuple encoded as nested pairs), and $\mathsf{r}^\mathsf{d}_\mathsf{a}$ is a name with $\Gamma(\mathsf{r}^\mathsf{d}_\mathsf{a}) = \mathrm{Low}$. However, this output is performed under some conditions. In particular, agent $\mathsf{a}[i]$ tests the tag ("2"), as well as that $\mathsf{snd}(\mathsf{snd}(\mathsf{snd}(t^\mathsf{d}_{\mathsf{adec}}))) = \mathsf{a}^\mathsf{d}[j]$. When typing this output, we have to consider the case where $t^\mathsf{d}_{\mathsf{adec}}$ is of type T_0, and the case where $t^\mathsf{d}_{\mathsf{adec}}$ is of type Low (relying on BREAK-SUM). The latter case is easily handled using rule FUN-LOW. To deal with the former one, we rely on the tag, and also on the test of the agent name, to show that the condition is always false, and thus the resulting output is the constant empty, of type Low. This check of the responder's identity is thus crucial to the typechecking of the protocol. This is not surprising, since the secrecy of nonce N_b famously does not hold on the flawed Needham-Schroeder protocol, where that test is missing [25] – as expected, we cannot typecheck it. In the end, the secrecy of N_b (as seen by A and B) can be established using type-checking.

A similar analysis is also performed on Mechanism 6 of [22].

Implementation. The implementation has been extended to deal with asymmetric encryption, and the two case studies above are available in the supplementary material of this publication. This extension was mostly straightforward and behaves as expected. The full heuristic (see Section VI-A) regarding the way the rule BREAK-SUM is applied is relevant for these case studies: applying the BREAK-SUM as soon as possible will not allow us to typecheck *e.g.* the second action of the responder (played by an honest agent with an honest agent) for the NSL protocol.

C. Discussion

The extension to asymmetric encryption nicely demonstrates the extensibility of our approach. Integrating the asymmetric encryption primitive in our type system did not require us to alter the main structure of the soundness proof, or to re-do it. Of course, some work was needed for arguments specific to asymmetric encryption (adding oracles to the evaluator, reduction of IND-CCA-2, *etc.*), but these points do not interfere with the parts of the soundness proof already written for

symmetric encryption. What made this possible is the fact that the IND-CCA-2 assumption fits nicely in our decomposition between restricted and base system. Indeed, as seen above, that assumption allowed us to remove the decryption operation, going from the base level to the restricted level, and then to hide secrets inside encrypted messages, at the restricted level. Although we do not formally prove or even state that claim, we believe that, more generally, the same would be true for other primitives: as long as their properties fit nicely into our decomposition, it should be possible to integrate them. For instance, we believe that the PRF assumption, postulating that a hash function is indistinguishable from a pseudo-random function, could be integrated in our framework.

The asymmetric encryption primitive highlights the importance of a design choice we made for public terms. In our system, the attacker cannot deduce the interpretation of a secret term (type High) from that of a public one (type Low), as specified in Definition 5. However, we do not require that he is indeed able to compute the value of public terms. Another approach, adopted by OWL [20], is to characterise public terms as those effectively computable by an attacker. This is of importance in the case of asymmetric encryption: does decrypting an adversarially computed (Low) message produce a plaintext already computable by the adversary? That property would be needed to type it Low (as rule ADEC does) with the alternative approach. However, it is not implied by IND-CCA-2 alone – another axiom is needed, known as Plaintext Awareness [7], which is not satisfied by all IND-CCA-2 cryptosystems. Although this assumption does not explicitly appear in [20], the authors confirmed that they indeed need it, while our approach does not.

It may seem surprising that we only analyse two protocols using asymmetric encryption. The reason is that many public-key protocols (*e.g.* other mechanisms in [22]) also use signatures, which is not yet supported by our type system.

Moreover, our system cannot currently establish the secrecy of N_a from the point of view of B in NSL. Indeed, the first message B receives could be forged by the attacker, so the second component of the decrypted message, *i.e.* N_a as seen by B, could be public – until B confirms it with A in later messages, which our type system cannot see. Although not entirely satisfactory, this limitation of our work is acceptable, considering our main goal is to help SQUIRREL users to prove secrecy properties. Agreement properties are already handled rather well by SQUIRREL, and in the case of NSL, the agreement property stating that A and B agree on the value of N_a at the end, would be sufficient to establish secrecy of N_a as seen by B, from the the secrecy of N_a as seen by A (property that can be established using our type system).

VIII. CONCLUSION

We have proposed a type system to establish non-deducibility of secrets, in the CCSA framework, supporting symmetric and asymmetric encryption primitives. We have proved our type system to be computationally sound, and we have applied it to a selection of case studies.

As discussed previously, our type system has some limitations regarding its scope which we plan to address in the future. A first aspect is the support of additional primitives, notably hash functions and signatures. Similarly to the extension to asymmetric encryption, this should only require rather local changes, as our approach is extensible. A second direction is to handle a larger class of properties. An interesting approach would be to take inspiration from symbolic type systems, by tagging types with events to express authentication guarantees – the actions from the CCSA logic could play that role. We also plan to study how to strengthen our type system to show forward secrecy, where a value must remain secret even if keys are corrupted later on. Dealing with key usability, where keys are exchanged and then used to encrypt secrets, would likely require finer-grained levels of secrecy: non-deducibility is indeed not sufficient for a key to be usable – a stronger notion of secrecy is required.

Finally, we envision being able to leverage properties or invariants proved in SQUIRREL to help the typechecking, *e.g.* regarding the order in which some actions are performed. This allows us to prune some branches of the typing tree, which would be helpful when considering *e.g.* forward secrecy.

REFERENCES

[1] M. Abadi, "Secrecy by typing in security protocols," *J. ACM*, vol. 46, no. 5, p. 749–786, sep 1999.

[2] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, "An Interactive Prover for Protocol Verification in the Computational Model," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 537–554.

[3] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, "Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols using the Squirrel Proof Assistant," in *Proceedings of the 35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*. IEEE, 2022, pp. 289–304.

[4] G. Bana and H. Comon-Lundh, "A Computationally Complete Symbolic Attacker for Equivalence Properties," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 2014, pp. 609–620.

[5] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-Aided Security Proofs for the Working Cryptographer," in *Proceedings of the Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011.*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, 2011, pp. 71–90.

[6] D. A. Basin, J. Dreier, and R. Sasse, "Automated Symbolic Proofs of Observational Equivalence," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*. ACM, 2015, pp. 1144–1155.

[7] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-Privacy in Public-Key Encryption," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 566–582.

[8] K. Bhargavan, A. Bichhawat, Q. H. Do, P. Hosseyni, R. Küsters, G. Schmitz, and T. Würtele, "DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code," in *Proceedings of the IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 2021, pp. 523–542.

[9] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, 2001, pp. 82–96.

[10] ——, "A Computationally Sound Mechanized Prover for Security Protocols," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA*. IEEE Computer Society, 2006, pp. 140–154.

[11] B. Blanchet and B. Smyth, "Automated reasoning for equivalences in the applied pi calculus with barriers," *Journal of Computer Security*, vol. 26, no. 3, pp. 367–422, 2018.

[12] C. Brzuska, A. Delignat-Lavaud, C. Fournet, K. Kohbrok, and M. Kohlweiss, "State Separation for Code-Based Game-Playing Proofs," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Part III*, ser. Lecture Notes in Computer Science, vol. 11274. Springer, 2018, pp. 222–249.

[13] M. Burrows, M. Abadi, and R. M. Needham, "A Logic of Authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18–36, 1990.

[14] J. A. Clark and J. Jacob, "On the Security of Recent Protocols," *Inf. Process. Lett.*, vol. 56, no. 3, pp. 151–155, 1995.

[15] V. Cortier, N. Grimm, J. Lallemand, and M. Maffei, "A Type System for Privacy Properties," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM, 2017, pp. 409–423.

[16] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Z. Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, "Implementing and Proving the TLS 1.3 Record Layer," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 463–482.

[17] A. Delignat-Lavaud, C. Fournet, B. Parno, J. Protzenko, T. Ramananandro, J. Bosamiya, J. Lallemand, I. Rakotonirina, and Y. Zhou, "A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer," in *Proceedings of the 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1162–1178.

[18] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[19] R. Focardi and M. Maffei, "Types for Security Protocols," *Formal Models and Techniques for Analyzing Security Protocols*, pp. 143–181, 2011, publisher: IOS Press.

[20] J. Gancher, S. Gibson, P. Singh, S. Dharanikota, and B. Parno, "Owl: Compositional Verification of Security Protocols via an Information-Flow Type System," in *Proceedings of the 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 1130–1147.

[21] IT Security techniques - Key management, "ISO/IEC 11770, Part 2: Mechanisms using symmetric techniques (third edition)," International Standard, Tech. Rep., 2018.

[22] ——, "ISO/IEC 11770, Part 3: Mechanisms using asymmetric techniques (fourth edition)," International Standard, Tech. Rep., 2021.

[23] P. Laud, "Handling Encryption in an Analysis for Secure Information Flow," in *Proceedings of Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003*, ser. Lecture Notes in Computer Science, P. Degano, Ed., vol. 2618. Springer, 2003, pp. 159–173.

[24] P. Laud and V. Vene, "A Type System for Computationally Secure Information Flow," in *Proceedings of the Fundamentals of Computation Theory*, M. Liśkiewicz and R. Reischuk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 365–377.

[25] G. Lowe, "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR," *Softw. Concepts Tools*, vol. 17, no. 3, pp. 93–102, 1996.

[26] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Proceedings of the Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, ser. LNCS, vol. 8044. Springer, 2013, pp. 696–701.

[27] G. Smith and R. Alpízar, "Secure information flow with random assignment and encryption," in *Proceedings of the 2006 ACM workshop on Formal methods in security engineering, FMSE 2006, Alexandria, VA, USA, November 3, 2006*. ACM, 2006, pp. 33–44.

APPENDIX A
BACKGROUND

We present here the elements of the background that we only introduce informally in Section III. In particular, we present the cryptographic games expressing the standard assumptions we make on the cryptographic primitives under study (see Section A-A), as well as the full syntax and semantics regarding the meta-logic. Indeed, for sake of clarity, we do not introduce macros allowing one to model states in Section III and we only provide an informal description of the meta-logic semantics. The interested reader will find its full formal description in Section A-B.

A. Cryptographic games

This appendix presents the cryptographic games expressing standard assumptions we make on the security of encryption primitives.

a) IND-CPA*:*

$\mathsf{G}^{\mathsf{indcpa},\beta}_{\mathcal{A}}(1^\eta,\rho)$
$k \leftarrow_\$ \{0,1\}^\eta$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{e},\mathcal{O}_\mathsf{LR}}(1^\eta,\rho')$
return β'

$\mathcal{O}_\mathsf{e}(m)$
$r \leftarrow_\$ \{0,1\}^\eta$
return $[\![\mathsf{senc}]\!]^{\mathbb{M}}(1^\eta,m,k,r)$

$\mathcal{O}_\mathsf{LR}(m_0,m_1)$
$r \leftarrow_\$ \{0,1\}^\eta$
return $[\![\mathsf{senc}]\!]^{\mathbb{M}}(1^\eta,m_\beta,k,r)$

$\mathcal{A}$ is allowed to call $\mathcal{O}_\mathsf{e}$ any number of times, and $\mathcal{O}_\mathsf{LR}$ only once. ρ' is the tape ρ, except the parts used by the encryption oracles and the key generation. The advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{indcpa}}_{\mathcal{A}}(\eta) = \left|\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},0}_{\mathcal{A}}(1^\eta,\rho)=1\right]-\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},1}_{\mathcal{A}}(1^\eta,\rho)=1\right]\right|.$$

b) INT-CTXT*:*

$\mathsf{G}^{\mathsf{intctxt}}_{\mathcal{A}}(1^\eta,\rho)$
$k \leftarrow_\$ \{0,1\}^\eta$
$\mathsf{L}_\mathsf{c} \leftarrow [\,]$
$c \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{e},\mathcal{O}_\mathsf{d}}(1^\eta,\rho')$
if $c \notin \mathsf{L}_\mathsf{c}$
  $\wedge\ [\![\mathsf{sdec}]\!]^{\mathbb{M}}(1^\eta,m,k) \neq [\![\mathsf{fail}]\!]^{\mathbb{M}}$
then return 1
else return 0.

$\mathcal{O}_\mathsf{e}(m)$
$r \leftarrow_\$ \{0,1\}^\eta$
$c \leftarrow [\![\mathsf{senc}]\!]^{\mathbb{M}}(1^\eta,m,k,r)$
$\mathsf{L}_\mathsf{c} \leftarrow \mathsf{append}(c,\mathsf{L}_\mathsf{c})$
return c

$\mathcal{O}_\mathsf{d}(c)$
return $[\![\mathsf{sdec}]\!]^{\mathbb{M}}(1^\eta,c,k)$

$\mathcal{A}$ is allowed to call $\mathcal{O}_\mathsf{e}$ and $\mathcal{O}_\mathsf{d}$ any number of times. ρ' is the tape ρ, except the parts used by the encryption oracle and the key generation. The advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{intctxt}}_{\mathcal{A}}(\eta) = \mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{intctxt}}_{\mathcal{A}}(1^\eta,\rho)=1\right].$$

c) IND-CCA-2*:*

$\mathsf{G}^{\mathsf{indcca},\beta}_{\mathcal{A}}(1^\eta,\rho)$
$sk \leftarrow_\$ \{0,1\}^\eta$
$pk \leftarrow \mathsf{pk}(sk)$
$ch \leftarrow \bot$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{d},\mathcal{O}_\mathsf{LR}}(pk,1^\eta,\rho')$
return β'

$\mathcal{O}_\mathsf{d}(c)$
if $ch = \bot \vee c \neq ch$ **then**
  return $[\![\mathsf{adec}]\!]^{\mathbb{M}}(1^\eta,c,sk)$
else
  return $\bot$

$\mathcal{O}_\mathsf{LR}(m_0,m_1)$
$r \leftarrow_\$ \{0,1\}^\eta$
$ch \leftarrow [\![\mathsf{aenc}]\!]^{\mathbb{M}}(1^\eta,m_\beta,sk,r)$
return c

$\mathcal{A}$ can call $\mathcal{O}_\mathsf{LR}$ only once and $\mathcal{O}_\mathsf{d}$ any number of times. Note that $\mathcal{O}_\mathsf{d}$ answers any query before $\mathcal{O}_\mathsf{LR}$ has been called, and only queries other than the challenge ciphertext afterwards. ρ' is the tape ρ, except for the parts used by the left-right oracle and the key generation. The advantage of $\mathcal{A}$ is

$$\mathsf{Adv}^{\mathsf{indcca}}_{\mathcal{A}}(\eta) = \left|\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcca},0}_{\mathcal{A}}(1^\eta,\rho)=1\right]-\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcca},1}_{\mathcal{A}}(1^\eta,\rho)=1\right]\right|.$$

B. Meta-logic

In order to study properties of protocols, [2] introduces the meta-logic level. Basically, a protocol execution model is introduced in the form of an *action system*. Meta-logic terms can contain macros that refer to messages produced in any action along the execution, *e.g.* $\mathsf{output}@\tau$ refers to the message output at time τ. Using these macros, meta-logic formulas can then express generic properties on protocol executions.

In order to allow reasoning on many sessions of protocols, the meta-logic introduces a notion of *indices*. Constants, names, and encryption randomness may take as arguments indices, that identify the protocol session they belong to.

1) Syntax: The syntax of the meta-logic consists of terms of three sorts, namely *index*, *timestamp* and *message*. Formally, we assume infinite sets $\mathcal{I}$, $\mathcal{X}$, $\mathcal{T}$ of index, message, and timestamp variables. We assume two symbols init, pred, used to represent the initial timestamp and a predecessor operation on timestamps. We also assume finite sets $\mathcal{N}$, $\mathcal{R}$, $\mathcal{C}$ of *indexed* names, random values, and constants: each of these symbols has an index arity. We denote by $\mathcal{C}^0$ (resp. $\mathcal{C}^\infty$) the subset of $\mathcal{C}$ of index arity 0 (resp. non-zero). We assume a finite set of function symbols $\mathcal{F}$ of index arity 0, but each symbol has a message arity $k \geq 0$. To distinguish index arguments from message arguments, we write them between brackets, *e.g.* $\mathsf{n}[i_1,\ldots,i_k]$ vs $\mathsf{f}(m_1,\ldots,m_l)$.

We assume a finite set of *macro* symbols $\mathcal{M}$ given with an index arity. This set is composed of five built-in symbols output, input, frame, exec, cond, each with index arity 0; and of a set $\mathcal{S}$ of *state symbols* of arbitrary arity. Finally we assume a finite set $\mathcal{A}$ of *action* symbols, given with their arity, used to denote protocol steps.

The only terms of sort index are index variables. Terms of sort timestamp and message are built as follows:

$$\begin{array}{rcl} T & ::= & \tau \mid \mathsf{A}[\vec{i}\,] \mid \mathsf{init} \mid \mathsf{pred}(T) \\ t & ::= & x \mid \mathsf{m}[\vec{i}\,]@T \mid \mathsf{n}[\vec{i}\,] \mid \mathsf{f}(\vec{t}\,) \mid \vec{i} =_\iota \vec{j} \end{array}$$

where $\tau \in \mathcal{T}$, $\mathsf{A} \in \mathcal{A}$, $x \in \mathcal{X}$, $\mathsf{m} \in \mathcal{M}$, $\mathsf{n} \in \mathcal{N} \cup \mathcal{R} \cup \mathcal{C}$, $\mathsf{f} \in \mathcal{F}$, $\vec{i}$ and $\vec{j}$ are vectors of indices, and $\vec{t}$ a vector of messages. We sometimes call *meta-terms* the meta-logic terms.

An *action* A is defined as:

$$\mathsf{A}[\vec{i}\,].(\phi_{\mathsf{A}[\vec{i}\,]}, o_{\mathsf{A}[\vec{i}\,]}, \{\mathsf{s}[\vec{j}\,] \leftarrow u_{\mathsf{A}[\vec{i}\,],\mathsf{s}[\vec{j}\,]} \mid \mathsf{s} \in \mathcal{S}, \vec{j} \in \mathcal{I}\})$$

where $\vec{i}$ is a vector of distinct indices in $\mathcal{I}$, understood in this notation as bound variables, $\phi_{\mathsf{A}[\vec{i}]}$, $o_{\mathsf{A}[\vec{i}]}$ are messages called the *condition* and *output*, and for any symbol $\mathsf{s} \in \mathcal{S}$, for any distinct indices $\vec{j}$ disjoint from $\vec{i}$ (also used as bound variables in this notation), $u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}$ is a message describing the *update* of state $\mathsf{s}[\vec{j}]$ at action $\mathsf{A}[\vec{i}]$.

A *protocol* $\mathcal{P} = (\mathcal{A}\mathsf{ct}, \prec)$ is composed of:

- a finite set $\mathcal{A}\mathsf{ct}$ of actions (one for each symbol in $\mathcal{A}$);
- and a partial order $\prec$ on terms of the form $\mathsf{A}[\vec{i}]$. $\prec$ must be invariant by alpha-renaming indices.

We require that $\mathcal{A}\mathsf{ct}$ contains a particular action init, with no indices, that is smaller than all others for $\prec$. Its condition is $\phi_{\mathsf{init}} = \mathsf{true}$, its output $o_{\mathsf{init}} = \mathsf{empty}$, and its update $\mathcal{U}_0 = \{\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{init},\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}\}$ provides an initial value for each state. $u_{\mathsf{init},\mathsf{s}[\vec{j}]}$ must be a macro-free term of sort message, with free variables in $\vec{j}$.

In addition, we require that in action $\mathsf{A}[\vec{i}]$, all timestamps refer to earlier actions:

- either $\mathsf{A}'[\vec{j}]$ such that $\mathsf{A}'[\vec{j}] \prec \mathsf{A}[\vec{i}]$;
- or $\mathsf{pred}(\mathsf{A}[\vec{i}])$ when $\mathsf{A} \neq \mathsf{init}$;
- or $\mathsf{A}[\vec{i}]$ itself as a parameter to input;
- or $\mathsf{A}[\vec{i}]$ itself as a parameter to a state macro $\mathsf{s} \in \mathcal{S}$ when the occurrence is in $o_{\mathsf{A}[\vec{i}]}$.

We demand that in action $\mathsf{A}[\vec{i}]$, all updates $\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}$ are of the form

$$u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]} = \mathsf{if}\ \vec{j} =_\iota \vec{i}\,' \ \mathsf{then}\ u\ \mathsf{else}\ \mathsf{s}[\vec{j}]@\mathsf{pred}(\mathsf{A}[\vec{i}]),$$

where $\vec{i}\,'$ is a subset of $\vec{i}$. Finally, we require that all randoms $\mathsf{r} \in \mathcal{R}$ appearing in $\mathsf{A}[\vec{i}]$ are applied to exactly $\vec{i}$, *i.e.* $\mathsf{r}[\vec{i}]$.

2) Semantics: The semantics of the meta-logic is defined by translating meta-logic terms into base-logic terms. Given a protocol $\mathcal{P}$, a trace model $\mathbb{T}$ of $\mathcal{P}$ is composed of

- a finite index domain $\mathcal{D}_\mathcal{I} \subseteq \mathbb{N}$;
- a timestamp domain $\mathcal{D}_\mathcal{T}$ containing undef, and elements of the form $\mathsf{A}[\vec{k}]$ where $\mathsf{A} \in \mathcal{A}$ and $\vec{k} \in \mathcal{D}_\mathcal{I}$;
- a total ordering $<_\mathcal{T}$ on $\mathcal{D}_\mathcal{T} \setminus \{\mathsf{undef}\}$, compatible with $\prec$ in the sense that: $\mathsf{A}[\vec{i}] \prec \mathsf{A}'[\vec{j}]$ implies $\mathsf{A}[\sigma(\vec{i})] <_\mathcal{T} \mathsf{A}'[\sigma(\vec{j})]$ for any $\sigma : \mathcal{I} \to \mathcal{D}_\mathcal{I}$;
- $\sigma_\mathcal{I} : \mathcal{I} \to \mathcal{D}_\mathcal{I}$, $\sigma_\mathcal{T} : \mathcal{T} \to \mathcal{D}_\mathcal{T}$ are mappings interpreting index and timestamp variables.

For a trace model $\mathbb{T}$, we define a predecessor function $\mathsf{pred}_\mathcal{T}$ as expected in accordance with $<_\mathcal{T}$, and we set $\mathsf{pred}_\mathcal{T}(\mathsf{init})$ to undef. Let $\mathbb{T}\{\tau \mapsto T\}$ denote the trace model $\mathbb{T}$ where $\sigma_\mathcal{T}$ is updated to map τ to T, and similarly for an update of $\sigma_\mathcal{I}$.

Given a protocol $\mathcal{P}$ and a trace model $\mathbb{T}$, we now define the translation from meta-logic terms to base-logic terms. We take an instance of the base logic such that $\mathcal{F}^\mathcal{B} = \mathcal{F}$, $\mathcal{N}^\mathcal{B} = \{\mathsf{n}_{\vec{k}} \mid \mathsf{n} \in \mathcal{N}, \vec{k} \in \mathcal{D}_\mathcal{I}\}$, and similarly for randoms $\mathcal{R}^\mathcal{B}$. We take $\mathcal{C}^\mathcal{B} = \mathcal{C}^0 \cup \bigcup_{c \in \mathcal{C}^\infty} \{\mathsf{c}_{\vec{k}} \mid \vec{k} \in \mathcal{D}_\mathcal{I}\}$, and we choose $\mathcal{X}^\mathcal{B} = \mathcal{X} \cup \mathcal{X}_\mathcal{M}$, where $\mathcal{X}_\mathcal{M} = \{x_{\mathsf{m}[\vec{k}]@T} \mid \mathsf{m} \in \mathcal{M}, \vec{k} \in \mathcal{D}_\mathcal{I}, T \in \mathcal{D}_\mathcal{T}\}$ is a set of variables that will be used as stand-ins for macros.

The translation of terms of sort timestamp is as expected, using $\sigma_\mathcal{T}$, $\sigma_\mathcal{I}$ and $\mathsf{pred}_\mathcal{T}$:

$$\overline{\tau}^\mathbb{T} = \sigma_\mathcal{T}(\tau), \quad \overline{\mathsf{A}[\vec{i}]}^\mathbb{T} = \mathsf{A}[\sigma_\mathcal{I}(\vec{i})], \quad \overline{\mathsf{pred}(T)}^\mathbb{T} = \mathsf{pred}_\mathcal{T}(\overline{T}^\mathbb{T}).$$

For terms t of sort message, we proceed in two steps. First, we define $\overline{t}^\mathbb{T}$ to be t where each macro is replaced with a variable that will be instantiated later with the body of the macro. Formally:

- $\overline{\mathsf{f}(\vec{t})}^\mathbb{T} = \mathsf{f}(\overline{\vec{t}}^\mathbb{T})$ for variables and function symbols;
- names, constants and randoms are translated by the appropriate base-level symbol: $\overline{\mathsf{n}[\vec{i}]}^\mathbb{T} = \mathsf{n}_{\sigma_\mathcal{I}(\vec{i})}$;
- index equalities are interpreted as booleans depending on the value of the indices: $\overline{i =_\iota j}^\mathbb{T} = \mathsf{true}$ if $\sigma_\mathcal{I}(i) = \sigma_\mathcal{I}(j)$, and false otherwise;
- $\overline{\mathsf{m}[\vec{i}]@T}^\mathbb{T} = x_{\mathsf{m}[\sigma_\mathcal{I}(\vec{i})]@\overline{T}^\mathbb{T}}$ for macros.

As a second step, we define how macro variables are instantiated. We simultaneously define $\theta^\mathbb{T}_\mathcal{P}$, and $(\cdot)^\mathbb{T}_\mathcal{P}$, as follows. For any meta-term t and trace model $\mathbb{T}$, $(t)^\mathbb{T}_\mathcal{P}$ is $\overline{t}^\mathbb{T}\theta^\mathbb{T}_\mathcal{P}$, and for any $\mathsf{m} \in \mathcal{M}, \vec{l} \in \mathcal{D}_\mathcal{I}, T \in \mathcal{D}_\mathcal{T}$:

- when $T = \mathsf{undef}$, $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{m}[\vec{l}]@T})$ is false if $\mathsf{m} \in \{\mathsf{cond}, \mathsf{exec}\}$ and empty otherwise;
- when $T = \mathsf{A}[\vec{k}]$ for an action

  $$\mathsf{A}[\vec{i}].(\phi_{\mathsf{A}[\vec{i}]}, o_{\mathsf{A}[\vec{i}]}, \{\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}, \vec{j} \in \mathcal{J}\}),$$

  denoting $\mathbb{T}' = \mathbb{T}\{\vec{i} \mapsto \vec{k}\}$, we let

  $$\begin{aligned}
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{output}@T}) &= (\mathsf{if}\ \phi_{\mathsf{A}[\vec{i}]}\mathsf{then}\ o_{\mathsf{A}[\vec{i}]}\ \mathsf{else}\ \mathsf{empty})^{\mathbb{T}'}_\mathcal{P} \\
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{cond}@T}) &= (\phi_{\mathsf{A}[\vec{i}]})^{\mathbb{T}'}_\mathcal{P} \\
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{m}[\vec{l}]@T}) &= (u_{\mathsf{A}[\vec{i}],\mathsf{m}[\vec{j}]})^{\mathbb{T}\{\vec{i}\mapsto\vec{k},\vec{j}\mapsto\vec{l}\}}_\mathcal{P}\ \text{if } \mathsf{m} \in \mathcal{S}.
  \end{aligned}$$

  If A is init, we let $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{input}@T}) = \theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{frame}@T}) = \mathsf{empty}$, and $\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{exec}@T}) = \mathsf{true}$. Otherwise:

  $$\begin{aligned}
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{input}@T}) &= (\mathsf{att}(\mathsf{frame}@\mathsf{pred}(\mathsf{A}[\vec{i}])))^{\mathbb{T}'}_\mathcal{P} \\
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{frame}@T}) &= (\langle \mathsf{exec}@\mathsf{A}[\vec{i}], \\
  &\qquad \langle \mathsf{if}\ \mathsf{exec}@\mathsf{A}[\vec{i}]\mathsf{then}\ \mathsf{output}@\mathsf{A}[\vec{i}] \\
  &\qquad \mathsf{else}\ \mathsf{empty}, \mathsf{frame}@\mathsf{pred}(\mathsf{A}[\vec{i}])\rangle\rangle)^{\mathbb{T}'}_\mathcal{P} \\
  -\theta^\mathbb{T}_\mathcal{P}(x_{\mathsf{exec}@T}) &= (\mathsf{cond}@T \wedge \mathsf{exec}@\mathsf{pred}(\mathsf{A}[\vec{i}]))^{\mathbb{T}'}_\mathcal{P}.
  \end{aligned}$$

Compared to [3], this translation is very similar, except that the expansion of the output macro explicitly involves the condition $\phi_{\mathsf{A}[\vec{i}]}$. This makes no difference when considering the frame macro, as that condition was already present there, but ensures we only consider the output of an action when that action can effectively be performed, which may provide useful information when typechecking.

Appendix B
Main result (with states)

In this section, we present our main result for stateful protocols. The addition of state macros in the meta-logic allowing one to model stateful protocols implies only very few changes, which are formally stated below.

First, we have to consider typing environment allowing one to give a type to states, and also consider two extra typing rules in order to type state macro, and also equality between

indices useful to express how to update a state. These rules, namely .., are given below.

We extend Definition 2 to allow state macros from $\mathcal{S}$ in Γ.

Definition 6 (typing environment)**.** *A* typing environment $(\Gamma; R)$ *is composed of*

- *a finite set of bindings* Γ *giving a message or key type* T *to some elements of* $\mathcal{X} \cup \mathcal{N} \cup \mathcal{S}$*;*
- *a finite set* $R \subseteq \mathcal{R}$ *of random symbols that can be used for encryptions.*

We say that Γ*, and by extension* $(\Gamma; R)$*, is* well-formed *if*

(i) Γ *does not contain multiple bindings for the same symbol;*
(ii) for $n \in \mathcal{N}$*,* $\Gamma(n)$ *is either* High*,* Low *or a key type;*
(iii) for $x \in \mathcal{X} \cup \mathcal{S}$*,* $\Gamma(x)$ *is a message type.*

We introduce two typing rules to type state macros and to handle indices equalities used in their semantics. These rules concerns macros and indices, so we add them to Figure 3:

$$\frac{\Gamma(\mathsf{s}) = \mathrm{T}}{\Gamma; R \vdash \mathsf{s}[\vec{i}]@T : \mathrm{T}}$$

$$\frac{}{\Gamma; R \vdash \vec{i} =_\iota \vec{j} : \mathrm{Bool}}$$

We also have to adapt Definition 3 to take into account states, and to ensure that terms stored in states are well-typed.

Definition 7 (well-typed protocols)**.** *Let* $(\Gamma; R)$ *be a well-formed typing environment that does not contain variables, and* $\mathcal{P}$ *a protocol. We say that* $\mathcal{P}$ *is* well-typed *in* $(\Gamma; R)$ *when for each action*

$$\mathsf{A}[\vec{i}].(\phi_{\mathsf{A}[\vec{i}]}, o_{\mathsf{A}[\vec{i}]}, \{\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{A}[\vec{i}]\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}\})$$

composing the protocol:

- $\Gamma; \emptyset \vdash \phi_{\mathsf{A}[\vec{i}]} : \mathrm{Bool}$*;*
- $\Gamma; R_{\mathsf{A}} \vdash \mathsf{if}\ \phi_{\mathsf{A}[\vec{i}]}\ \mathsf{then}\ o_{\mathsf{A}[\vec{i}]}\ \mathsf{else}\ \mathsf{empty} : \mathrm{Low}$*;*
- *for each* $\mathsf{s} \in \mathcal{S}$*,* $\Gamma; R^{\mathsf{s}}_{\mathsf{A}} \vdash u_{\mathsf{A}[\vec{i}]\mathsf{s}[\vec{j}]} : \Gamma(\mathsf{s})$*.*

for some sets $\{R_{\mathsf{A}}\}_{\mathsf{A} \in \mathcal{A}}$*,* $\{R^{\mathsf{s}}_{\mathsf{A}}\}_{\mathsf{A} \in \mathcal{A}, \mathsf{s} \in \mathcal{S}}$ *that form a partition of* R*. We impose that* $R^{\mathsf{s}}_{\mathsf{init}} = \emptyset$ *for any* $\mathsf{s} \in \mathcal{S}$*.*

Once, this has been modified, the main result, Theorem 1, stated in Section IV, remains the same. Of course, the proof of this result is a little bit more involved when considering protocols featuring states. Proofs of this result are detailed in Appendices C-E.

Appendix C
Restricted system for messages (base logic)

In this section, we establish a first soundness result, for a subset of the type system, with the aim of typing terms from the base logic.

A. Restricted type sytem

In the context of type-checking base-logic terms, we will consider environments that store not only the encryption randoms that may be used, but also which terms they may encrypt.

Definition 8. *A mapping environment, denoted* $(\Gamma; \mathcal{R})$*, is composed of*

- *a finite set of bindings* Γ *giving a message or key type* T *to some elements of* $\mathcal{X}^{\mathcal{B}} \cup \mathcal{N}^{\mathcal{B}}$*;*
- *a finite-domain mapping* $\mathcal{R}$*, associating to some random symbol* $r \in \mathcal{R}^{\mathcal{B}}$ *a pair* (m, k) *of ground base-logic terms.*

The mapping environment $(\Gamma; \mathcal{R})$ *is* well-formed *if* Γ *is, and* $\mathcal{R}$ *does not contain multiple bindings for the same symbol.*

The type system we consider in this section, called the *restricted type system*, is composed of the rules given in Figure 6. These are basically the rules from Figure 1, modified to use a mapping typing environment, and to remove all indices used by meta-terms.

A *typing judgement* w.r.t. this restricted system is an expression of the form $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ where $(\Gamma; \mathcal{R})$ is a well-formed mapping environment, t is a ground base-logic term, and T is a message type.

Basically, all specific rules that handle destructors (*e.g.* Fst, Sdec, If-False) are absent from the restricted type system – though the generic rules for functions can still be applied to them. This way, typechecking a term in the restricted type system requires typechecking all its subterms. This is formally stated in Lemma 4. The destructor rules, that allow to avoid type-checking some subterms, are handled separately in the next section.

We say that $(\Gamma; \mathcal{R})$ is *well-typed* if it is well-formed, and for all r, denoting $\mathcal{R}(r) = (m, k)$, we have $\Gamma(k) = \mathrm{SK[T]}$ for some T such that $\Gamma; \mathcal{R} \vdash_r m : \mathrm{T}$.

B. Properties of the restricted system

We start with a few simple lemmas that express guarantees given by the restricted type system.

Lemma 1. *There does not exist a message type* T *such that* $\mathrm{T} \leq \mathrm{High}$*, and* $\mathrm{T} \leq \mathrm{Low}$*.*

Proof. Let T be a message type. We have that T is of the form $\mathrm{T}_1 \times \ldots \times \mathrm{T}_n$ where each T_i is not a product. In case $\mathrm{T} \leq \mathrm{High}$, we have that $\mathrm{T}_{i_0} = \mathrm{High}$ for some i_0. In case $\mathrm{T} \leq \mathrm{Low}$, we have that $\mathrm{T}_i \in \{\mathrm{Low}, \mathrm{Bool}\} \cup \{\mathrm{Cst}^\star_{\mathsf{c}} \mid \mathsf{c} \in \mathscr{C}^0 \cup \mathscr{C}^\infty\}$ for each $i \in \{1, \ldots, n\}$. Therefore, we conclude that these two cases are incompatible, and therefore a message type T can not be such that $\mathrm{T} \leq \mathrm{High}$ and $\mathrm{T} \leq \mathrm{Low}$. □

Lemma 2. *Let* t *be a ground term and* $(\Gamma; \mathcal{R})$ *a well-formed mapping environment such that* $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ *for some type* T*.*

- *For any variable* x*,* $\Gamma \setminus \{x\}; \mathcal{R} \vdash_r t : \mathrm{T}$*, where* $\Gamma \setminus \{x\}$ *is the environment obtained from* Γ *by removing the binding for* x *(if there was one).*
- *For any random symbol* r *that does not appear in* t*,* $\Gamma; \mathcal{R} \setminus \{r\} \vdash_r t : \mathrm{T}$*, where* $\mathcal{R} \setminus \{r\}$ *is defined similarly.*

Proof. We prove both properties by induction on the type derivation of $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$. If that proof is reduced to a single application of rule Name, Cst-0, or Cst-∞, both claims clearly hold. In all other cases, the last rule of the derivation has premises judgements of the form $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}'$, where t' is

$$\frac{\Gamma(n) = \mathrm{T}}{\Gamma; \mathcal{R} \vdash_r n : \mathrm{T}}\ \textsc{Name}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t : \mathrm{T} \qquad \mathrm{T} \leq \mathrm{T}'}{\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}'}\ \textsc{Sub-Typing}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_1 : \mathrm{T} \qquad \Gamma; \mathcal{R} \vdash_r t_2 : \mathrm{T}'}{\Gamma; \mathcal{R} \vdash_r \langle t_1, t_2 \rangle : \mathrm{T} \times \mathrm{T}'}\ \textsc{Pair}$$

$$\frac{\mathsf{c} \in \mathscr{C}^0}{\Gamma; \mathcal{R} \vdash_r \mathsf{c} : \mathrm{Cst}^0_{\mathsf{c}}}\ \textsc{Cst-0} \qquad \frac{\mathsf{c} \in \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}'} \quad \mathsf{c}' \in \mathscr{C}^{\mathcal{B}} \setminus \mathscr{C}^0}{\Gamma; \mathcal{R} \vdash_r \mathsf{c} : \mathrm{Cst}^{\infty}_{\mathsf{c}'}}\ \textsc{Cst-}\infty$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t : \mathrm{Msg}}{\Gamma; \mathcal{R} \vdash_r \mathsf{zeros}(t) : \mathrm{Low}}\ \textsc{Zeros}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{Low} \quad \text{for } i = 1, \ldots, n}{\Gamma; \mathcal{R} \vdash_r \mathsf{f}(t_1, ..., t_n) : \mathrm{Low}}\ \textsc{Fun-Low}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{Msg} \quad \text{for } i = 1, \ldots, n}{\Gamma; \mathcal{R} \vdash_r \mathsf{f}(t_1, ..., t_n) : \mathrm{Msg}}\ \textsc{Fun-Msg}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t : \mathrm{T} \qquad \Gamma(k) = \mathrm{SK}[\mathrm{T}] \qquad \mathcal{R}(r) = (t, k)}{\Gamma; \mathcal{R} \vdash_r \mathsf{senc}(t, k, r) : \mathrm{Low}}\ \textsc{Senc}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_1 : \mathrm{Msg} \qquad \Gamma; \mathcal{R} \vdash_r t_2 : \mathrm{Msg}}{\Gamma; \mathcal{R} \vdash_r t_1 = t_2 : \mathrm{Bool}}\ \textsc{Eq}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_1 : \mathrm{High} \qquad \Gamma; \mathcal{R} \vdash_r t_2 : \mathrm{Low}}{\Gamma; \mathcal{R} \vdash_r t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}}\ \textsc{Eq-False}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_1 : \mathrm{Cst}^0_c \qquad \Gamma; \mathcal{R} \vdash_r t_2 : \mathrm{Cst}^0_c}{\Gamma; \mathcal{R} \vdash_r t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{true}}}\ \textsc{Eq-True-Cst}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{Cst}^\star_{c_i} \qquad \text{with } i = 1, 2 \text{ and } c_1 \neq c_2}{\Gamma; \mathcal{R} \vdash_r t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}}\ \textsc{Eq-False-Cst}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t : \mathrm{Bool} \qquad \Gamma; \mathcal{R} \vdash_r t_i : \mathrm{T} \text{ with } i = 1, 2}{\Gamma; \mathcal{R} \vdash_r \mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathrm{T}}\ \textsc{If}$$

Fig. 6: Typing rules for the restricted system

a subterm of t. Since t is ground, t' is ground as well, and if r does not appear in t, then it does not in t' either. We then conclude by applying the induction hypothesis to t'. □

Lemma 3. *Let t be a ground term and $(\Gamma; \mathcal{R})$ a well-formed mapping environment such that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ for some type T. Consider $\Gamma' \supseteq \Gamma$ and $\mathcal{R}' \supseteq \mathcal{R}$ such that $(\Gamma'; \mathcal{R}')$ is well-formed. Then $\Gamma'; \mathcal{R}' \vdash_r t : \mathrm{T}$.*

Proof. This lemma is easily proved by induction on the derivation of $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$. Indeed, in all possible typing rules, extending (in a well-formed way) the environments does not render any of the premises false, and we conclude by applying the induction hypothesis to all premises that are typing judgements. □

Lemma 4. *Let t be a ground term and $(\Gamma; \mathcal{R})$ a well-formed environment, and Π be a typing derivation of $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ for some type T. Each subterm in t is either:*

- *a name k assigned to a key type in Γ,*
- *a random $r \in \mathrm{dom}(\mathcal{R})$, or*
- *a term t' for which there exists Π' a subtree of Π witnessing $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}'$ for some message type T'.*

Proof. Induction on the typing tree. Axioms only apply to atomic terms. Sub-typing does not modify the term. All other rules have in requirement the typing of each direct subterm. □

Lemma 6 is useful to prove that the restricted type system guarantees secrecy of terms of type High. It states that when a term typed High appears inside a public term (*i.e.* of type Low), then it is occurs in a context that will preserve its secrecy. This context can be an encryption with a valid key, a zeros that hides everything but its length, or an equality that leak at most a bit of information. The following lemma is an intermediate step useful to prove Lemma 6.

Lemma 5. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, and t a ground term such that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ and $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}'$. It is not possible that both $\mathrm{T} \leq \mathrm{High}$ and $\mathrm{T}' \leq \mathrm{Low}$.*

Proof. Let Π (resp. Π') be the typing derivation for judgement $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ (resp. $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}'$). We show by induction on $\mathsf{size}(\Pi) + \mathsf{size}(\Pi')$, where size denotes the number of nodes in the derivation, that having both $\mathrm{T} \leq \mathrm{High}$ and $\mathrm{T}' \leq \mathrm{Low}$ leads to a contradiction. We distinguish several cases.

- Π *ends with the rule* Sub-Typing. There exists T'' such that $\mathrm{T}'' \leq \mathrm{T} \leq \mathrm{High}$ and $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}''$, with a proof tree Π'' such that $\mathsf{size}(\Pi'') \leq \mathsf{size}(\Pi)$. Applying our induction hypothesis on Π'' and Π' leads to a contradiction.
- Π' *ends with the rule* Sub-Typing. There exists T'' such that $\mathrm{T}'' \leq \mathrm{T}' \leq \mathrm{Low}$ and $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}''$ with a proof tree Π'' such that $\mathsf{size}(\Pi'') \leq \mathsf{size}(\Pi')$. Applying our induction hypothesis on Π'' and Π leads to a contradiction.
- Π *ends with the rule* Name. We have already eliminated the case where Π' ends with the rule Sub-Typing. Thus, Π' necessarily ends with the rule Name too, and therefore $\mathrm{T} = \mathrm{T}'$. We conclude by Lemma 1.
- Π *ends with the rule* If. In such a case, as Π' does not end with rule Sub-Typing, Π' ends with the rule If or Fun-Low. The case Fun-Msg is not possible, as $\mathrm{Msg} \not\leq \mathrm{Low}$. In both cases, we have $t = \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$, and we conclude relying on the induction hypothesis on *e.g.* $i = 1$.
- Π *ends with the ule* Pair. In such a case, as Π' does not end with rule Sub-Typing, Π' ends with the rule Pair or Fun-Low. The case Fun-Msg is not possible as $\mathrm{Msg} \not\leq \mathrm{Low}$. In both cases, we have $t = \langle t_1, t_2 \rangle$, and $\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{T}_i$ for $i = 1, 2$. Moreover, we have $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2 \leq \mathrm{High}$, and thus $\mathrm{T}_i \leq \mathrm{High}$ for some $i_0 \in \{1, 2\}$. We conclude by applying the induction hypothesis to the left subtrees if $i_0 = 1$, or the right subtrees if $i_0 = 2$.

The other cases (*i.e.* Π ending with Cst-0, Cst-∞, Zeros, Fun-Msg, Senc, Eq, Eq-False, Eq-True-Cst, Eq-False-Cst) are not possible, as they cannot lead to a type T such that $\mathrm{T} \leq \mathrm{High}$. □

Lemma 6. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, and t_L be a ground term such that $\Gamma; \mathcal{R} \vdash_r t_L : \mathrm{Low}$. If $t_L = C[t_H]$, for a t_H such that $\Gamma; \mathcal{R} \vdash_r t_H : \mathrm{High}$ and a*

context C (i.e. a term with a single hole), then C has one of the following forms:

- $C = C_1[\mathsf{senc}(C_2, k, r)]$, *for some contexts* C_1, C_2, *with* $\mathcal{R}(r) = (C_2[t_H], k)$, *and* $\Gamma(k) = \mathrm{SK}[\mathrm{T}]$ *for some* T;
- $C = C_1[\mathsf{zeros}(C_2)]$ *for some* C_1, C_2;
- $C = C_1[C_2 = t]$ *for some contexts* C_1, C_2 *and term* t;
- $C = C_1[t = C_2]$ *for some* C_1, C_2, t.

Proof. We show in fact a slightly stronger version of this lemma: the same property holds as soon as $\Gamma; \mathcal{R} \vdash_r t_L : \mathrm{T}$ for some $\mathrm{T} \leq \mathrm{Low}$. We prove this generalised property by induction on the type derivation Π of that judgement.

By Lemma 5, we know that t_L itself cannot be given type High. Thus, only strict subterms t_H of t_L, *i.e.* non-trivial contexts C, need to be considered. We distinguish several cases for the last rule in Π.

- Rules Name, Cst-0, Cst-∞. We have that t_L is a name or constant, and does not have any strict subterm.
- Rules Senc, Zeros, Eq, Eq-False, Eq-True-Cst, Eq-False-Cst. Any strict subterm of t_L is contained in a context satisfying the claim. For instance, in the case of rule Zeros, $t_L = \mathsf{zeros}(t'_L)$, and any non-trivial context C around a strict subterm is of the form $C = \mathsf{zeros}(C')$.
- Rule Sub-Typing. We have that $\Gamma; \mathcal{R} \vdash_r t_L : \mathrm{T}'$ for some $\mathrm{T}' \leq \mathrm{T} \leq \mathrm{Low}$, with a proof shorter than Π. We conclude immediately by applying the induction hypothesis.
- Rule Fun-Low, and If are similar. We only present Fun-Low. In that case, $t_L = \mathsf{f}(t_L^1, \ldots, t_L^n)$, and the strict subterm t_H is located in one of the t_L^i, *i.e.* $C = \mathsf{f}(t_L^1, \ldots, t_L^{i-1}, C', t_L^{i+1}, t_L^n)$ for some C', such that $t_L^i = C'[t_H]$. One of the rule's premises provides a proof (shorter than Π) that $\Gamma; \mathcal{R} \vdash_r t_L^i : \mathrm{Low}$. By the induction hypothesis, we obtain that C' has an adequate form, and therefore C as well.
- Rule Pair: then $t_L = \langle t_L^1, t_L^2 \rangle$, and $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$ for some t_L^1, t_L^2, T_1, T_2, and additionally $\Gamma; \mathcal{R} \vdash_r t_L^i : \mathrm{T}_i$ for $i = 1, 2$, with proofs shorter than Π. Assume w.l.o.g. that t_H occurs in t_L^1 (the other case is similar), *i.e.* $C = \langle C', t_L^2 \rangle$ for some C' such that $C'[t_H] = t_L^1$. Since $\mathrm{T}_1 \times \mathrm{T}_2 \leq \mathrm{Low}$, we have $\mathrm{T}_1 \leq \mathrm{Low}$. By applying the induction hypothesis to t_L^1, we get that C' has the correct form, and thus so does C.

Note that Π can not end with Fun-Msg as $\mathrm{Msg} \not\leq \mathrm{Low}$. □

Finally, the following lemma establishes how typing handles logical connectors.

Lemma 7. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment. If $\Gamma; \mathcal{R} \vdash_r b_i : \mathrm{Bool}$ for $i \in \{1, 2\}$, then we have that:*

$$\Gamma; \mathcal{R} \vdash_r t : \mathrm{Bool} \text{ when } t \in \{\neg b_1, b_1 \wedge b_2, b_1 \vee b_2\}.$$

Proof. Application of rules If, Cst-0 (for true and false), and Sub-Typing (with $\mathrm{Cst}^0_{\mathsf{false}}$, $\mathrm{Cst}^0_{\mathsf{true}} \leq \mathrm{Bool}$). □

C. Evaluator

Definition 9. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, $\mathbb{M}$ a computational model, and t a ground base-logic term. The* evaluator *of t in $\mathbb{M}, \Gamma, \mathcal{R}$, denoted $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)$, is a PPTM with access to η, the adversarial random tape ρ_{a}, and two oracles $\mathcal{O}_{\mathsf{name}}, \mathcal{O}_{\mathsf{enc}}$. For better legibility, we consider them as a single oracle $\mathcal{O} = \{\mathcal{O}_{\mathsf{name}}, \mathcal{O}_{\mathsf{enc}}\}$, when there is no ambiguity as to which oracle is called. The oracle $\mathcal{O}$ has access to the entire random tape $\rho = (\rho_{\mathsf{h}}, \rho_{\mathsf{a}})$. It is defined inductively as follows:*

- *for any name $n \in \mathcal{N}^{\mathcal{B}}$, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(n)(1^\eta, \rho_{\mathsf{a}})$ calls $\mathcal{O}_{\mathsf{name}}(\text{“}n\text{”})$ and returns its answer;*
- *for any random $r \in \mathcal{R}^{\mathcal{B}}$, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(r)(1^\eta, \rho_{\mathsf{a}})$ fails;*
- *if k is a name such that $\Gamma(k) = \mathrm{SK}[\mathrm{T}]$ for some T, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{senc}(m, k, r))(1^\eta, \rho_{\mathsf{a}})$, first computes $p = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(m)(1^\eta, \rho_{\mathsf{a}})$, and then calls $\mathcal{O}_{\mathsf{enc}}(p, \text{“}k\text{”}, \text{“}r\text{”})$ and returns its answer.*
- *else for any function $\mathsf{f} \in \mathcal{F}^{\mathcal{B}} \cup \{\mathsf{att}\}$ of arity ℓ, and all terms $t_1, \ldots, t_\ell$, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{f}(t_1, \ldots, t_\ell))(1^\eta, \rho_{\mathsf{a}})$ first computes the arguments $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t_i)(1^\eta, \rho_{\mathsf{a}})$ for all i, and then calls $[\![\mathsf{f}]\!]^{\mathbb{M}}$ on the resulting values.*
- *If any of the recursive calls to $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}$ or the oracle calls to $\mathcal{O}$ fail, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}})$ fails as well.*

Note that we write “r” to denote that the symbol *r (assuming an encoding of symbols as bitstrings) is submitted to the oracle, and not the bitstring $[\![r]\!]^{\mathbb{M}}(1^\eta, \rho)$ interpreting its value.*

The oracle $\mathcal{O} = \{\mathcal{O}_{\mathsf{name}}, \mathcal{O}_{\mathsf{enc}}\}$ is defined as follows:

- *for any name n, $\mathcal{O}_{\mathsf{name}}(\text{“}n\text{”})$ returns $[\![n]\!]^{\mathbb{M}}(1^\eta, \rho)$ if n is bound to a message type in Γ, and fails otherwise;*
- *for any random r, key k, and bitstring m, $\mathcal{O}_{\mathsf{enc}}(m, \text{“}k\text{”}, \text{“}r\text{”})$ checks whether $\mathcal{R}(r) = (m', k)$ for some m' such that $m = [\![m']\!]^{\mathbb{M}}(1^\eta, \rho)$. If so, it returns $[\![\mathsf{senc}]\!]^{\mathbb{M}}(m, [\![k]\!]^{\mathbb{M}}(1^\eta, \rho), [\![r]\!]^{\mathbb{M}}(1^\eta, \rho))$, and it fails otherwise.*

It can easily be seen by induction on t that for any well-formed $(\Gamma; \mathcal{R})$, any $\mathbb{M}$ and any t, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}})$ either fails or returns $[\![t]\!]^{\mathbb{M}}(1^\eta, \rho)$. The evaluator succeeds if it never needs to make forbidden calls to $\mathcal{O}$, *i.e.* if t uses keys and randoms in the way prescribed by $(\Gamma; \mathcal{R})$.

Definition 10. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, and t a ground base-logic term. We say that t is* evaluable *in $(\Gamma; \mathcal{R})$ if for any computational model $\mathbb{M}$, $\mathbb{P}_\rho\left[\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}}) = [\![t]\!]^{\mathbb{M}}(1^\eta, \rho)\right] \in \mathsf{ow}(\eta)$.*

Lemma 8. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, t a ground base-logic term such that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$ for some message type T. We have that t is evaluable in $(\Gamma; \mathcal{R})$.*

Proof. We prove this lemma by induction on the derivation of the judgement $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$, and examine each possible last rule in it. Most cases are easily proved. The interesting cases are rules Name and Senc.

In the Name case, $t = n$ is a name, and $\Gamma(n) = \mathrm{T}$ is a message type. For any η, ρ, the evaluator $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(n)(1^\eta, \rho_{\mathsf{a}})$ thus calls its oracle $\mathcal{O}_{\mathsf{name}}$, which answers with $[\![n]\!]^{\mathbb{M}}(1^\eta, \rho)$, and the claim holds.

In the SENC case, we have $t = \mathsf{senc}(m,k,r)$, $\mathrm{T} = \mathrm{Low}$, $\Gamma;\mathcal{R} \vdash_r m : \mathrm{T}'$, $\Gamma(k) = \mathrm{SK}[\mathrm{T}']$, and $\mathcal{R}(r) = (m,k)$. To run $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{senc}(m,k,r))(1^\eta,\rho_\mathsf{a})$, the evaluator first executes $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(m)(1^\eta,\rho_\mathsf{a})$. By the induction hypothesis, with overwhelming probability, this succeeds and returns $p = [\![m]\!]^{\mathbb{M}}(1^\eta,\rho)$. The evaluator then calls $\mathcal{O}(p,\text{“}k\text{”},\text{“}r\text{”})$. The oracle's check succeeds, and $\mathcal{O}_{\mathsf{enc}}$ correctly computes the encryption of m. □

D. Type interpretation

In order to state and prove our soundness result regarding the restricted typing system, we first have to define what it means for a term to be in the interpretation of a type T. We start with some preliminary definitions, and notations:

- $t \mathrel{\tilde{\in}} S$, where S is a finite set of ground base-logic terms, when for any computational model $\mathbb{M}$,
$$\mathbb{P}_\rho\left[\exists t' \in S.\, [\![t]\!]^{\mathbb{M}}(1^\eta,\rho) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\right] \in \mathsf{ow}(\eta).$$
- $t \approx t'$ when $t \mathrel{\tilde{\in}} \{t'\}$;

Note that it follows immediately from these definitions that $\approx$ is an equivalence relation, and that for any function symbol $f \in \mathcal{F}^{\mathcal{B}}$, if $t_1 \approx t'_1, \ldots, t_n \approx t'_n$, then $f(t_1,\ldots,t_n) \approx f(t'_1,\ldots,t'_n)$ (recall that functions are interpreted as deterministic PPTMs). Moreover, for any ground t_1,t_2,t'_1,t'_2 such that $t_1 \approx t'_1$ and $t_2 \approx t'_2$, if $t_1 \blacktriangleright t_2$ then $t'_1 \blacktriangleright t'_2$.

Definition 5. *Given a well-formed $(\Gamma;\mathcal{R})$, a ground base-logic term t, and a message type T, we say that t* is in the interpretation of T w.r.t. $(\Gamma;\mathcal{R})$, *denoted by $\Gamma;\mathcal{R} \models t : \mathrm{T}$, when there exists $t' \approx t$ such that $\Gamma;\mathcal{R} \vdash_r t' : \mathrm{T}$, and:*

- *if $\mathrm{T} = \mathrm{Cst}^0_{\mathsf{c}}$: $t \approx \mathsf{c}$;*
- *if $\mathrm{T} = \mathrm{Cst}^\infty_{\mathsf{c}}$: $t \mathrel{\tilde{\in}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$;*
- *if $\mathrm{T} = \mathrm{Bool}$: $t \mathrel{\tilde{\in}} \{\mathsf{true},\mathsf{false}\}$;*
- *if $\mathrm{T} = \mathrm{Msg}$ or $\mathrm{T} = \mathrm{Low}$: no further condition;*
- *if $\mathrm{T} = \mathrm{High}$: for all t_L, if $\Gamma;\mathcal{R} \models t_L : \mathrm{Low}$ then $t_L \blacktriangleright t$;*
- *if $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$: $\Gamma;\mathcal{R} \models \mathsf{fst}(t) : \mathrm{T}_1$, and $\Gamma;\mathcal{R} \models \mathsf{snd}(t) : \mathrm{T}_2$;*
- *if $\mathrm{T} = \mathrm{T}_1 + \mathrm{T}_2$: either $\Gamma;\mathcal{R} \models t : \mathrm{T}_1$, or $\Gamma;\mathcal{R} \models t : \mathrm{T}_2$, or there exist b, t_1, t_2 such that $t' = \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$, $\Gamma;\mathcal{R} \models b : \mathrm{Bool}$, and $\Gamma;\mathcal{R} \models t_i : \mathrm{T}_i$ for $i = 1,2$.*

The following lemma allows us to choose another term with the same semantics (up to negligible probability) as a term t.

Lemma 9. *Let $(\Gamma;\mathcal{R})$ be a well-formed mapping environment, and t,t' be two ground base-logic terms such that $t \approx t'$. If $\Gamma;\mathcal{R} \models t : \mathrm{T}$, then $\Gamma;\mathcal{R} \models t' : \mathrm{T}$.*

Proof. If $\Gamma;\mathcal{R} \models t : \mathrm{T}$, there exists t'' such that $t \approx t''$, that satisfies the conditions listed in Definition 5. As $\approx$ is transitive, we also have $t' \approx t''$, and thus $\Gamma;\mathcal{R} \models t' : \mathrm{T}$. □

Lemma 10. *Let $(\Gamma;\mathcal{R})$ be a well-formed mapping environment, t a ground base-logic term, and T, T′ message types such that $\mathrm{T} \leq \mathrm{T}'$. If $\Gamma;\mathcal{R} \models t : \mathrm{T}$, then $\Gamma;\mathcal{R} \models t : \mathrm{T}'$.*

Proof. We prove this lemma by induction on the subtyping proof tree deriving $\mathrm{T} \leq \mathrm{T}'$. Let us first note that by definition of $\models$, there exists t' such that $t \approx t'$ and $\Gamma;\mathcal{R} \vdash_r t' : \mathrm{T}$. Thus, by subtyping, $\Gamma;\mathcal{R} \vdash_r t' : \mathrm{T}'$, which takes care of the first condition required for $\Gamma;\mathcal{R} \models t : \mathrm{T}'$. We now show the additional conditions, that depend on T′. To do so, we distinguish several cases, depending on the last subtyping rule applied to get $\mathrm{T} \leq \mathrm{T}'$.

- Case of the rules for which $\mathrm{T}' \in \{\mathrm{Low},\mathrm{Msg}\}$: then no further condition is required, and the claim holds.
- Case of the rules for which $\mathrm{T}' = \mathrm{Bool}$, and thus $\mathrm{T} \in \{\mathrm{Cst}^0_{\mathsf{false}},\mathrm{Cst}^0_{\mathsf{true}}\}$. Then, since $\Gamma;\mathcal{R} \models t : \mathrm{T}$, $t \approx \mathsf{false}$ (or true), and the condition is satisfied.
- Case of the sum rule: $\mathrm{T}' = \mathrm{T}_1 + \mathrm{T}_2$, and $\mathrm{T} = \mathrm{T}_i$ for some $i \in \{1,2\}$. Then $\Gamma;\mathcal{R} \models t : \mathrm{T}_i$, which shows the condition required for $\Gamma;\mathcal{R} \models t : \mathrm{T}_1 + \mathrm{T}_2$.
- Case of the pair rule: $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$ and $\mathrm{T}' = \mathrm{T}'_1 \times \mathrm{T}'_2$, with $\mathrm{T}_i \leq \mathrm{T}'_i$ for $i = 1,2$. Since $\Gamma;\mathcal{R} \models t : \mathrm{T}$, by Definition 5, we know that $\Gamma;\mathcal{R} \models \mathsf{fst}(t) : \mathrm{T}_1$ and also $\Gamma;\mathcal{R} \models \mathsf{snd}(t) : \mathrm{T}_2$. Thus by the induction hypothesis, $\Gamma;\mathcal{R} \models \mathsf{fst}(t) : \mathrm{T}'_1$ and $\Gamma;\mathcal{R} \models \mathsf{snd}(t) : \mathrm{T}'_2$, which shows the condition for $\Gamma;\mathcal{R} \models t : \mathrm{T}'_1 \times \mathrm{T}'_2$ is satisfied.
- Case of the rules where $\mathrm{T}' = \mathrm{High}$ and $\mathrm{T} = \mathrm{High} \times \mathrm{Msg}$ or $\mathrm{T} = \mathrm{Msg} \times \mathrm{High}$. These two cases are similar, we only detail the first one. Let t_L be a ground term such that $\Gamma;\mathcal{R} \models t_L : \mathrm{Low}$. Showing that $t_L \blacktriangleright t$ will prove the claim. By contradiction, assume there exists (for some model $\mathbb{M}$) a PPTM $\mathcal{A}$ such that $\mathbb{P}_\rho\left[\mathcal{A}([\![t_L]\!]^{\mathbb{M}}(1^\eta,\rho),1^\eta,\rho_a) = [\![t]\!]^{\mathbb{M}}(\eta,\rho)\right]$ is non-negligible (in η). By composing $\mathcal{A}$ with the machine interpreting fst in $\mathbb{M}$, we obtain a machine $\mathcal{A}'$ that computes $\mathsf{fst}(t)$ from t_L with good probability: $\mathbb{P}_\rho\left[\mathcal{A}'([\![t_L]\!]^{\mathbb{M}}(1^\eta,\rho),1^\eta,\rho_a) = [\![\mathsf{fst}(t)]\!]^{\mathbb{M}}(\eta,\rho)\right]$ is non-negligible.
  On the other hand, by assumption we have that:
  $$\Gamma;\mathcal{R} \models t : \mathrm{High} \times \mathrm{Msg}.$$
  Hence, by Definition 5, $\Gamma;\mathcal{R} \models \mathsf{fst}(t) : \mathrm{High}$. This implies that $t_L \blacktriangleright \mathsf{fst}(t)$, which is contradicted by the existence of the machine $\mathcal{A}'$.

Only the reflexivity and transitivity cases remains. The reflexivity case trivially holds. In the transitivity case, there exists T″ such that $\mathrm{T} \leq \mathrm{T}''$ and $\mathrm{T}'' \leq \mathrm{T}'$. Then, by applying the induction hypothesis twice, we get first $\Gamma;\mathcal{R} \models t : \mathrm{T}''$, and then $\Gamma;\mathcal{R} \models t : \mathrm{T}'$, which concludes the proof. □

Lemma 11. *Let $(\Gamma;\mathcal{R})$ be a well-formed mapping environment, t a ground base-logic term, and T a message type.*
$$\textit{If } \Gamma;\mathcal{R} \models t : \mathrm{T} + \mathrm{T}, \textit{ then } \Gamma;\mathcal{R} \models t : \mathrm{T}.$$

Proof. We prove this lemma by induction on type T. By Definition 5, since $\Gamma;\mathcal{R} \models t : \mathrm{T} + \mathrm{T}$, two cases are possible. In the first case, $\Gamma;\mathcal{R} \models t : \mathrm{T}$, and the claim trivially holds. Now, we consider the second case, i.e. there exist b', t'_1, and t'_2 such that:

- $t \approx \mathsf{if}\ b'\ \mathsf{then}\ t'_1\ \mathsf{else}\ t'_2$,
- $\Gamma;\mathcal{R} \vdash_r \mathsf{if}\ b'\ \mathsf{then}\ t'_1\ \mathsf{else}\ t'_2 : \mathrm{T} + \mathrm{T}$,
- $\Gamma;\mathcal{R} \models b' : \mathrm{Bool}$, and $\Gamma;\mathcal{R} \models t'_i : \mathrm{T}$ for $i = 1,2$.

From that last point, we get that there exist b, t_1, t_2 such that $b \approx b'$, $t_i \approx t'_i$, $\Gamma;\mathcal{R} \vdash_r b : \mathrm{Bool}$, and $\Gamma;\mathcal{R} \vdash_r t_i : \mathrm{T}$ for

$i = 1, 2$. Thus, $t \approx$ if b then t_1 else t_2, and by applying rule IF, we get $\Gamma; \mathcal{R} \vdash_r$ if b then t_1 else $t_2 : \mathrm{T}$. Note, in addition, that by Lemma 9, we have $\Gamma; \mathcal{R} \models b : \mathrm{Bool}$, and $\Gamma; \mathcal{R} \models t_i : \mathrm{T}$ for $i = 1, 2$.

To finish the proof that $\Gamma; \mathcal{R} \models t : \mathrm{T}$, we must show additional conditions, depending on T. We therefore distinguish several possible cases.

- If $\mathrm{T} \in \{\mathrm{Low}, \mathrm{Msg}\}$: in that case we have nothing more to prove.
- If $\mathrm{T} = \mathrm{Cst}^0_{\mathsf{c}}$ (resp. $\mathrm{T} = \mathrm{Cst}^\infty_{\mathsf{c}}$) for some $\mathsf{c} \in \mathscr{C}^{\mathcal{B},0}$ (resp. $\mathsf{c} \in \mathcal{C}$), then, since $\Gamma; \mathcal{R} \models t'_i : \mathrm{T}$ for $i = 1, 2$, we have by definition $t'_i \approx \mathsf{c}$ (resp. $t'_i \mathrel{\tilde{\in}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$) for $i = 1, 2$, and therefore $t \approx$ if b' then t'_1 else $t'_2 \approx \mathsf{c}$ (resp. $t \mathrel{\tilde{\in}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$).
- If $\mathrm{T} = \mathrm{High}$: Then we have $\Gamma; \mathcal{R} \models t'_i : \mathrm{High}$ for $i = 1, 2$. We must show that $\Gamma; \mathcal{R} \models t : \mathrm{High}$. Consider a term t_L such that $\Gamma; \mathcal{R} \models t_L : \mathrm{Low}$. Showing that $t_L \blacktriangleright t$ will prove the claim. Let $\mathbb{M}$ be a model and $\mathcal{A}$ be an attacker. For an arbitrary η, we define:
  - $m_A \stackrel{\mathsf{def}}{=} \mathcal{A}([\![t_L]\!]^{\mathbb{M}}(1^\eta, \rho), 1^\eta, \rho_a)$;
  - $\mathsf{true}_\eta \stackrel{\mathsf{def}}{=} [\![\mathsf{true}]\!]^{\mathbb{M}}(1^\eta)$, and $\mathsf{false}_\eta \stackrel{\mathsf{def}}{=} [\![\mathsf{false}]\!]^{\mathbb{M}}(1^\eta)$;
  - $m_x \stackrel{\mathsf{def}}{=} [\![x]\!]^{\mathbb{M}}(\eta, \rho)$ for $x = t, t_1, t_2, b$.

  We have to prove that $\mathbb{P}_\rho[m_A = m_t] \in \mathsf{negl}(\eta)$. We will decompose this event according to the value of m_b:

$$\begin{aligned}
&\mathbb{P}_\rho[m_A = m_t] \\
=\ & \mathbb{P}_\rho[m_A = m_t \wedge m_b = \mathsf{true}_\eta] \\
&+ \mathbb{P}_\rho[m_A = m_t \wedge m_b = \mathsf{false}_\eta] \\
&+ \mathbb{P}_\rho[m_A = m_t \wedge m_b \notin \{\mathsf{true}_\eta, \mathsf{false}_\eta\}] \\
=\ & \mathbb{P}_\rho[m_A = m_{t_1} \wedge m_b = \mathsf{true}_\eta] \\
&+ \mathbb{P}_\rho[m_A = m_{t_2} \wedge m_b = \mathsf{false}_\eta] \\
&+ \mathbb{P}_\rho[m_A = m_t \wedge m_b \notin \{\mathsf{true}_\eta, \mathsf{false}_\eta\}] \\
\leq\ & \mathbb{P}_\rho[m_A = m_{t_1}] + \mathbb{P}_\rho[m_A = m_{t_2}] \\
&+ \mathbb{P}_\rho[m_b \notin \{\mathsf{true}_\eta, \mathsf{false}_\eta\}]
\end{aligned}$$

  By definition of $\Gamma; \mathcal{R} \models b : \mathrm{Bool}$, $b \mathrel{\tilde{\in}} \{\mathsf{true}, \mathsf{false}\}$, and so, $\mathbb{P}_\rho[m_b \notin \{\mathsf{true}_\eta, \mathsf{false}_\eta] \in \mathsf{negl}(\eta)$. Likewise, $\Gamma; \mathcal{R} \models t_i : \mathrm{High}$ implies $t_L \blacktriangleright t_i$ and thus $\mathbb{P}_\rho[m_A = m_{t_i}] \in \mathsf{negl}(\eta)$. We conclude that $\mathbb{P}_\rho[m_A = m_t] \in \mathsf{negl}(\eta)$, and so $t_L \blacktriangleright t$.
- If $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$, then we have $\Gamma; \mathcal{R} \models t'_i : \mathrm{T}_1 \times \mathrm{T}_2$ for $i = 1, 2$. Thus, by definition, for $i = 1, 2$, we have $\Gamma; \mathcal{R} \models \mathsf{fst}(t'_i) : \mathrm{T}_1$ and $\Gamma; \mathcal{R} \models \mathsf{snd}(t'_i) : \mathrm{T}_2$. Therefore, by definition of $\models$, we have that:
  - $\Gamma; \mathcal{R} \models$ if b' then $\mathsf{fst}(t'_1)$ else $\mathsf{fst}(t'_2) : \mathrm{T}_1 + \mathrm{T}_1$, and
  - $\Gamma; \mathcal{R} \models$ if b' then $\mathsf{snd}(t'_1)$ else $\mathsf{snd}(t'_2) : \mathrm{T}_2 + \mathrm{T}_2$.

  Above, we take terms overwhelmingly equal to $\mathsf{fst}(t'_i)$ and $\mathsf{snd}(t'_i)$ to ensure syntactic typing in the restricted system. This is not an issue as we know that $\Gamma; \mathcal{R} \models \mathsf{fst}(t'_i) : \mathrm{T}_1$ and $\Gamma; \mathcal{R} \models \mathsf{snd}(t'_i) : \mathrm{T}_2$. Then, by the induction hypothesis, we deduce that:
  - $\Gamma; \mathcal{R} \models$ if b' then $\mathsf{fst}(t'_1)$ else $\mathsf{fst}(t'_2) : \mathrm{T}_1$, and
  - $\Gamma; \mathcal{R} \models$ if b' then $\mathsf{snd}(t'_1)$ else $\mathsf{snd}(t'_2) : \mathrm{T}_2$.

  Moreover, since $t \approx$ if b' then t'_1 else t'_2, we have that $\mathsf{fst}(t) \approx$ if b' then $\mathsf{fst}(t'_1)$ else $\mathsf{fst}(t'_2)$ (and similarly for snd). Using Lemma 9, this means that $\Gamma; \mathcal{R} \models \mathsf{fst}(t) : \mathrm{T}_1$, and similarly for snd. Therefore, $\Gamma; \mathcal{R} \models t : \mathrm{T}_1 \times \mathrm{T}_2$.
- If $\mathrm{T} = \mathrm{T}_1 + \mathrm{T}_2$: that case is the most involved. We know that $\Gamma; \mathcal{R} \models t'_i : \mathrm{T}_1 + \mathrm{T}_2$ for $i = 1, 2$. Therefore, for each $i = 1, 2$, two situations are possible. Either *(i)* there exists $j_i \in \{1, 2\}$ such that $\Gamma; \mathcal{R} \models t'_i : \mathrm{T}_{j_i}$; or *(ii)* there exist b_i, t^1_i, t^2_i such that $t'_i \approx$ if b_i then t^1_i else t^2_i, $\Gamma; \mathcal{R} \models b_i : \mathrm{Bool}$, $\Gamma; \mathcal{R} \models t^1_i : \mathrm{T}_1$, and $\Gamma; \mathcal{R} \models t^2_i : \mathrm{T}_2$.

  Consider first the case where t'_1 and t'_2 are both in situation *(i)* with $j_1 = j_2$, *i.e.* $\Gamma; \mathcal{R} \models t'_i : \mathrm{T}_1$ for $i = 1, 2$ (the case T_2 is similar). Thus, by definition of $\models$, we have $\Gamma; \mathcal{R} \models$ if b' then t'_1 else $t'_2 : \mathrm{T}_1 + \mathrm{T}_1$, provided there exists a term overwhelmingly equal to that term which typechecks in the restricted system. That condition holds, as there are such equivalent typable terms for t'_1, t'_2, and b' to which we can apply rules SUB-TYPING and IF. Thus, using Lemma 9, we have that $\Gamma; \mathcal{R} \models t : \mathrm{T}_1 + \mathrm{T}_1$. By the induction hypothesis, we get that $\Gamma; \mathcal{R} \models t : \mathrm{T}_1$, and the condition for $\Gamma; \mathcal{R} \models t : \mathrm{T}_1 + \mathrm{T}_2$ is satisfied.

  In each other case, we prove $\Gamma; \mathcal{R} \models t : \mathrm{T}_1 + \mathrm{T}_2$ by finding terms t''_1, t''_2, b'' such that $\Gamma; \mathcal{R} \models b'' : \mathrm{Bool}$, $\Gamma; \mathcal{R} \models t''_i : \mathrm{T}_i$ for $i = 1, 2$, and $t \approx$ if b'' then t''_1 else t''_2. Indeed, if we have such terms, we can obtain by definition of $\models$ terms b''', t'''_1, t'''_2 that satisfy the same conditions and also typecheck in the restricted system. We can then use rules IF and SUB-TYPING to show that if b''' then t'''_1 else t'''_2 typechecks with type $\mathrm{T}_1 + \mathrm{T}_2$, which together with the other properties proves the claim. The choice of t''_1, t''_2, b'' depends on which situation applies to t_1, t_2.
  - Case *(i)* for both, with $\Gamma; \mathcal{R} \models t'_1 : \mathrm{T}_1$ and $\Gamma; \mathcal{R} \models t'_2 : \mathrm{T}_2$: we simply take $t''_1 = t'_1$, $t''_2 = t'_2$, and $b'' = b'$.
  - Case *(i)* for both, with $\Gamma; \mathcal{R} \models t'_1 : \mathrm{T}_2$ and $\Gamma; \mathcal{R} \models t'_2 : \mathrm{T}_1$: we take $t''_1 = t'_2$, $t''_2 = t'_1$, and $b'' = \neg b'$. We know that $\Gamma; \mathcal{R} \vdash_r b : \mathrm{Bool}$, and thus by Lemma 7, we have $\Gamma; \mathcal{R} \vdash_r \neg b : \mathrm{Bool}$. This allows us to deduce that $\Gamma; \mathcal{R} \models b'' : \mathrm{Bool}$ as $b'' \approx \neg b$, and the other conditions clearly hold.
  - Case *(ii)* for both: We take $t''_1 =$ if b then t^1_1 else t^1_2, $t''_2 =$ if b then t^2_1 else t^2_2, and $b'' = (b \wedge b_1) \vee (\neg b \wedge b_2)$. By checking each possible boolean value of b, b_1 and b_2, we find that $t \approx$ if b'' then t''_1 else t''_2. Relying on Lemma 7 to ensure the existence of an equivalent term that typechecks in the restricted system, we deduce that $\Gamma; \mathcal{R} \models b'' : \mathrm{Bool}$. In addition, following the same reasoning as in the "both (i) with $j_1 = j_2$" case earlier, we can show that $\Gamma; \mathcal{R} \models t''_i : \mathrm{T}_i + \mathrm{T}_i$ for $i = 1, 2$. Using the induction hypothesis, we thus have $\Gamma; \mathcal{R} \models t''_i : \mathrm{T}_i$, which concludes this case.
  - Case *(i)* for one and *(ii)* for the other one. All possible sub-cases are similar, we only detail the case where t'_1 is in situation *(ii)*, and $\Gamma; \mathcal{R} \models t'_2 : \mathrm{T}_1$. We take $t''_1 =$ if b then t^1_1 else t_2, $t''_2 = t^2_1$, and $b'' = (b \wedge b_1) \vee \neg b$. We then conclude similarly to the previous case. By checking each possible value of b and b_1, we can see that $t \approx$ if b'' then t''_1 else t''_2. By Lemma 7, we have $\Gamma; \mathcal{R} \models b'' : \mathrm{Bool}$. We already know that $\Gamma; \mathcal{R} \models t^2_1 : \mathrm{T}_2$. As before, we can show that $\Gamma; \mathcal{R} \models t''_1 : \mathrm{T}_1 + \mathrm{T}_1$.

By induction hypothesis, we thus have $\Gamma;\mathcal{R} \models t_1'' : \mathrm{T}_1$, which concludes the proof.

□

E. Some lemmas to deal with cryptography

Definition 11. *Let t and c be two ground base-logic terms with c of form $\mathsf{senc}(m,k,r)$. We let $\delta_c(t)$ denote the term obtained as follows:*

- $\delta_c(t) = t$ *when* $t \in \mathcal{N}^{\mathcal{B}} \cup \mathcal{C}^{\mathcal{B}} \cup \mathcal{R}^{\mathcal{B}}$;
- $\delta_c(\mathsf{senc}(m,k,r)) = \mathsf{senc}(\mathsf{zeros}(m),k,r)$; *and*
- $\delta_c(\mathsf{f}(t_1,..,t_n)) = \mathsf{f}(\delta_c(t_1),...,\delta_c(t_n))$ *otherwise.*

Definition 12. *Let $(\Gamma;\mathcal{R})$ be a well-typed mapping environment, and c a term of form $\mathsf{senc}(m,k,r)$ such that $\mathcal{R}(r) = (m,k)$.*

- *$\delta_c(\Gamma)$ denotes the mapping equal to Γ except for names mapped to types of form $\mathrm{SK[T]}$ that are mapped to $\mathrm{SK[Msg]}$.*
- *$\delta_c(\mathcal{R})$ denotes the mapping with the same domain as $\mathcal{R}$ and such that:*
  - *$\delta_c(\mathcal{R})(r) = (\mathsf{zeros}(m),k)$; and otherwise*
  - *$\delta_c(\mathcal{R})(r') = (\delta_c(t'),k')$ when $\mathcal{R}(r') = (t',k')$.*

Note that in $\delta_c(\Gamma)$, we replace types of the form $\mathrm{SK[T]}$ with $\mathrm{SK[Msg]}$, even for keys not used in c. These keys are used to encrypt messages of type T in $(\Gamma;\mathcal{R})$. Since $\mathrm{T} \leq \mathrm{Msg}$, we can use them to encrypt the same messages in $(\delta_c(\Gamma);\delta_c(\mathcal{R}))$.

Lemma 12. *Let $(\Gamma;\mathcal{R})$ be a well-typed mapping environment, t a base-logic term, and T a type such that $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T}$. Let c be a ground base-logic term of the form $\mathsf{senc}(m,k,r)$ and such that $\mathcal{R}(r) = (m,k)$. Then:*

- *$(\delta_c(\Gamma);\delta_c(\mathcal{R}))$ is a well-typed mapping environment;*
- *$\delta_c(\Gamma);\delta_c(\mathcal{R}) \vdash_r \delta_c(t) : \mathrm{T}$.*

Proof. We prove this lemma by induction on the typing derivation of $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T}$. All cases are straightforward, except those that may apply to $\mathsf{senc}(m,k,r)$, *i.e.* rules Senc, Fun-Low, and Fun-Msg. These last two cases are excluded as $r \in \mathrm{dom}(\mathcal{R}) \subseteq \mathcal{R}^{\mathcal{B}}$ cannot be given a message type.

In the case of rule Senc, we have $t = \mathsf{senc}(m,k,r)$, $\mathrm{T} = \mathrm{Low}$, $\Gamma(k) = \mathrm{SK[T']}$, $\mathcal{R}(r) = (m,k)$, and $\Gamma;\mathcal{R} \vdash_r m : \mathrm{T}'$. By induction hypothesis, we obtain $\delta_c(\Gamma);\delta_c(\mathcal{R}) \vdash_r \delta_c(m) : \mathrm{T}'$. Note that $\delta_c(m) = m$ as c is not a subterm of m. Hence $\delta_c(\Gamma);\delta_c(\mathcal{R}) \vdash_r \mathsf{zeros}(m) : \mathrm{Msg}$ by rules Sub-Typing and Zeros. We then conclude with an application of Senc. □

Lemma 13. *Consider two ground (base logic) terms t, t', a well-typed mapping environment $(\Gamma;\mathcal{R})$ such that $\Gamma;\mathcal{R} \vdash_r t : \mathrm{Msg}$ and $\Gamma;\mathcal{R} \vdash_r t' : \mathrm{Msg}$. Consider a term $c = \mathsf{senc}(m,k,r)$, such that $\mathcal{R}(r) = (m,k)$. Then:*

$$\delta_c(t) \blacktriangleright \delta_c(t') \text{ implies } t \blacktriangleright t'.$$

Proof. Consider a computational model $\mathbb{M}$ and a PPTM $\mathcal{A}$ that, given access to the adversarial tape ρ_{a} and to t, tries to compute t'. We have to show that $\mathbb{P}_{\rho}\left[\mathcal{A}(1^\eta,\rho_{\mathsf{a}},[\![t]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\right]$ is negligible in η.

The structure of the proof is as follows. We first establish a useful relation between the evaluators from Definition 9 for $(\Gamma;\mathcal{R})$ and $(\delta_c(\Gamma);\delta_c(\mathcal{R}))$. We show how t and $\delta_c(t)$ can be computed by the same evaluator. Finally, we construct an adversary $\mathcal{B}$ against Ind-Cpa, that uses these evaluators to simulate $\mathcal{A}$ on either t (on the left) or $\delta_c(t)$ (on the right), and checks if it manages to compute t' (resp. $\delta_c(t')$). By assumption, it likely fails on the right, and by the Ind-Cpa property, $\mathcal{B}$ must give with good probability the same answer on both sides, showing that $\mathcal{A}$ (on the left) has a low probability of computing t' from t.

Let us call respectively $\mathcal{O}_0$ and $\mathcal{O}_2$ the oracles given to the evaluators $\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}$, $\mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})}$ when evaluating in $(\Gamma;\mathcal{R})$ and $(\delta_c(\Gamma);\delta_c(\mathcal{R}))$. It is clear from Definition 9 that $\mathcal{E}_{\mathbb{M},\Gamma,\mathcal{R}}$ itself does not directly depend on $\mathcal{R}$ or Γ, but rather on the domain of Γ and on which names are given key types – which are the same for Γ and $\delta_c(\Gamma)$. The difference between evaluating a term in $(\Gamma;\mathcal{R})$ and $(\delta_c(\Gamma);\delta_c(\mathcal{R}))$ thus only lies in the oracle: we have that: $\mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})} = \mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\Gamma,\mathcal{R}}$. The two oracles $\mathcal{O}_0$ and $\mathcal{O}_2$ only differ in that $\mathcal{O}_0$ accepts to encrypt using r only the interpretation of m with the key k, while $\mathcal{O}_2$ only accepts to encrypt the interpretation of $\mathsf{zeros}(m)$ with key k.

Let us finally consider an additional oracle $\mathcal{O}_1$, which is similar to $\mathcal{O}_0$, except that it answers query $\mathcal{O}_1(m_0,\text{“}k_0\text{”},\text{“}r\text{”})$ by ensuring that $k_0 = k$ and $m_0 = [\![m]\!]^{\mathbb{M}}(1^\eta,\rho)$, and if so returns the encryption of $[\![\mathsf{zeros}]\!]^{\mathbb{M}}(m_0)$ with r, k. All other queries are answered like $\mathcal{O}_0$. When evaluating a term t_0, the evaluator $\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}(t_0)$ explores the term, and when reaching an encryption calls $\mathcal{O}_1$. $\mathcal{O}_1$ computes the corresponding ciphertext, except if the encryption was c, in which case it encrypts $\mathsf{zeros}(m)$: that is exactly the behaviour of the “normal” evaluator $\mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\Gamma,\mathcal{R}}$ when called on $\delta_c(t_0)$. Therefore, for any t_0, $\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}(t_0) = \mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\Gamma,\mathcal{R}}(\delta_c(t_0)) = \mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})}(\delta_c(t_0))$.

We know by assumption that t, t' are typable in $(\Gamma;\mathcal{R})$. Thus, by Lemma 8, they are both evaluable by $\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}$, *i.e.*

$$\mathbb{P}_{\rho}\left[\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_{\mathsf{a}}) = [\![t]\!]^{\mathbb{M}}(1^\eta,\rho_{\mathsf{a}})\right] \in \mathsf{ow}(\eta),$$

and similarly for t'. Besides, by Lemma 12, $\delta_c(t)$ and $\delta_c(t')$ are both typable in $(\delta_c(\Gamma);\delta_c(\mathcal{R}))$. Then, also by Lemma 8, $\delta_c(t)$ and $\delta_c(t')$ are evaluable by $\mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})}$, *i.e.* by $\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}$ (using the property above):

$$\mathbb{P}_{\rho}\left[\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_{\mathsf{a}}) = [\![\delta_c(t)]\!]^{\mathbb{M}}(1^\eta,\rho_{\mathsf{a}})\right] \in \mathsf{ow}(\eta),$$

and similarly for t'.

We construct a (PPTM) adversary $\mathcal{B}(1^\eta,\rho')$ against the Ind-Cpa game (see Appendix), where ρ' is w.l.o.g. seen as $(\rho'_{\mathsf{h}},\rho_{\mathsf{a}})$, ρ'_{h} being the part of the random tape ρ_{h} not associated with the symbol k or any encryption randomness symbol r' associated with k through $\mathcal{R}$. The machine $\mathcal{B}$ will internally simulate the execution of the evaluator $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ (depending on β) to compute the interpretations of some terms. That machine uses an oracle $\mathcal{O}_\beta$, which $\mathcal{B}$ simulates as follows:

- when called on a name “n”: fail if n is a key symbol in Γ, otherwise return the corresponding part of the random tape ρ'_{h}.

- when called on m_0, "k_0", "r_0" for a $k_0 \neq k$: check that k_0 is associated to r_0 in $\mathcal{R}$, and fail if not. If so, read the parts of ρ'_{h} corresponding to r_0, k_0, and use these values to encrypt m_0.
- when called on m_0, "k", "r_0", for a $r_0 \neq r$: check that k is associated to r_0 in $\mathcal{R}$, and fail if not. If so, call the encryption oracle from the IND-CPA game to encrypt m_0 and return the result. Record that result, to return it again if the same query is made later. This record persists across different runs of $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$.
- when called on m_0, "k", "r": call the left-right encryption oracle from the IND-CPA game on $(m_0, [\![\mathsf{zeros}]\!]^{\mathbb{M}}(m_0))$, and return the result. Again, record it so it can be reproduced later if needed.

When $\mathcal{O}_0$ (on the left) and $\mathcal{O}_1$ (on the right) succeed, they are accurately simulated, allowing $\mathcal{B}$ to simulate $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$. Note that this may not be the case when $\mathcal{O}_0$ fails: when simulating the $\mathcal{O}_0(m_0, "k", "r_0")$ call, $\mathcal{B}$ does not check that m_0 is the correct message, and may thus succeed while $\mathcal{O}_0$ does not. Later on we only need the simulation to be correct in the case $\mathcal{O}_0$ succeeds, so this is not an issue.

The machine $\mathcal{B}$ then proceeds as follows: it computes $m_\beta = \mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho)$ and $m'_\beta = \mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}(t')(1^\eta,\rho)$, and then simulates $\mathcal{A}$ on m_β (using its own ρ_{a} to simulate $\mathcal{A}$'s). The machine $\mathcal{B}$ obtains $\mathcal{A}(1^\eta, \rho_{\mathsf{a}}, m_\beta)$. If that value is equal to m'_β, $\mathcal{B}$ returns 0, otherwise (or if any of the evaluators fail at some point) it returns 1.

Consider first $\mathcal{B}$'s execution on the left side. Take any ρ such that $\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_{\mathsf{a}})$ successfully computes $[\![t]\!]^{\mathbb{M}}(1^\eta,\rho)$, and similarly for t', and at the same time $\mathcal{A}(1^\eta, \rho_a, [\![t]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)$. For such a ρ, $\mathcal{B}$ simulates $\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}$, and then $\mathcal{A}$, and thus $\mathcal{B}$ obtains in the end $[\![t']\!]^{\mathbb{M}}(1^\eta,\rho)$. The machine $\mathcal{B}$ thus returns 0. Therefore, we have that:

$$\begin{aligned}
&\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},0}_{\mathcal{B}}(1^\eta,\rho)=0\right]\\
\geq\ &\mathbb{P}_\rho\Big[\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_{\mathsf{a}}) = [\![t]\!]^{\mathbb{M}}(1^\eta,\rho)\\
&\quad\wedge \mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}(t')(1^\eta,\rho_{\mathsf{a}}) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\\
&\quad\wedge \mathcal{A}(1^\eta,\rho_a,[\![t]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\Big].
\end{aligned}$$

Since both evaluators have an overwhelming probability of success, we can deduce that

$$\begin{aligned}
&\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},0}_{\mathcal{B}}(1^\eta,\rho)=0\right] \geq\\
&\quad\mathbb{P}_\rho\left[\mathcal{A}(1^\eta,\rho_a,[\![t]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\right] - \mathsf{negl}(\eta).
\end{aligned}$$

Following a similar reasoning on the right, we obtain that

$$\begin{aligned}
&\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},1}_{\mathcal{B}}(1^\eta,\rho)=1\right] \geq\\
&\mathbb{P}_\rho\left[\mathcal{A}(1^\eta,\rho_a,[\![\delta_c(t)]\!]^{\mathbb{M}}(1^\eta,\rho)) \neq [\![\delta_c(t')]\!]^{\mathbb{M}}(1^\eta,\rho)\right] - \mathsf{negl}(\eta).
\end{aligned}$$

By assumption, $\delta_c(t) \blacktriangleright \delta_c(t')$, and thus that last probability is overwhelming. Thus, $\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},1}_{\mathcal{B}}(1^\eta,\rho)=0\right] \in \mathsf{negl}(\eta)$.

If $\mathbb{P}_\rho\left[\mathcal{A}(1^\eta,\rho_a,[\![t]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta,\rho)\right]$ was non-negligible, then by the inequality above, $\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},0}_{\mathcal{B}}(1^\eta,\rho)=0\right]$ would be as well. Therefore, we have that

$$\left|\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},1}_{\mathcal{B}}(1^\eta,\rho)=0\right] - \mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcpa},0}_{\mathcal{B}}(1^\eta,\rho)=0\right]\right|$$

is non-negligible which means that $\mathcal{B}$ has a non-negligible advantage in the IND-CPA game. Thus $\mathcal{A}$ has a negligible probability to compute t' from t, which concludes the proof. □

Definition 13. *Let t be a ground base-logic term, and k a name. Let $\delta_k(t)$ denote the term obtained from t as follows:*

- $\delta_k(t) = t$ *when* $t \in \mathcal{N}^{\mathcal{B}} \cup \mathcal{C}^{\mathcal{B}} \cup \mathcal{R}^{\mathcal{B}}$;
- $\delta_k(\mathsf{senc}(m,k,r)) = \mathsf{senc}(\mathsf{zeros}(\delta_k(m)),k,r)$ *if* $r \in \mathcal{R}^{\mathcal{B}}$; *and*
- $\delta_k(\mathsf{f}(t_1,\ldots,t_n)) = \mathsf{f}(\delta_k(t_1),\ldots,\delta_k(t_n))$ *otherwise.*

Definition 14. *Let $(\Gamma;\mathcal{R})$ be a well-typed mapping environment and k a name such that $\Gamma(k) = \mathrm{SK[T]}$ for some T.*

- $\delta_k(\Gamma)$ *denotes the mapping equal to Γ except for names mapped to types of form $\mathrm{SK[T]}$ that are mapped to $\mathrm{SK[Msg]}$.*
- $\delta_k(\mathcal{R})$ *denotes the mapping with the same domain as $\mathcal{R}$ defined by:*
  - $\delta_k(\mathcal{R})(r) = (\mathsf{zeros}(\delta_k(t)),k)$ *when* $\mathcal{R}(r) = (t,k)$*; and otherwise*
  - $\delta_k(\mathcal{R})(r) = (\delta_k(t),k')$ *when* $\mathcal{R}(r) = (t,k')$.

Lemma 14. *Let $(\Gamma;\mathcal{R})$ be a well-typed mapping environment, and t, t' two ground base-logic terms such that $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T}$ and $\Gamma;\mathcal{R} \vdash_r t' : \mathrm{T}'$ for some T, T'. Consider a name k such that $\Gamma(k) = \mathrm{SK[T]}$ for some T. Then:*

- $(\delta_k(\Gamma);\delta_k(\mathcal{R}))$ *is a well-typed mapping environment;*
- $\delta_k(\Gamma);\delta_k(\mathcal{R}) \vdash_r \delta_k(t) : \mathrm{T}$.
- $\delta_k(t) \blacktriangleright \delta_k(t')$ *implies* $t \blacktriangleright t'$.

Proof. Let $c_1,\ldots,c_\ell$ be the sequence of all (pairwise distinct) ciphertexts encrypted using k and a random symbol from $\mathrm{dom}(\mathcal{R})$ in t, t', and $\mathcal{R}$. We note $c_i = \mathsf{senc}(p_i,k,r_i)$. We number these ciphertexts in such a way that when $i<j$, c_i is not a subterm of c_j (*i.e.* of p_j). For any subterm m occurring in t, t', and $\mathcal{R}$, $\delta_k(m)$ is the term obtained from m by inserting zeros before each plaintext at each position where one of the c_i is present. Starting from m, the same result is obtained by doing so for each instance of c_1, then of c_2, c_3 and so on, *i.e.* applying successively $\delta_{c_1},\ldots,\delta_{c_\ell}$. Indeed, at each step, when applying δ_{c_j}, all ciphertexts c_i $(i<j)$ that have already been replaced do not appear as subterms of c_j, and therefore do not interfere with the replacement of c_j.

Let $\delta_i = \delta_{c_i} \circ \cdots \circ \delta_{c_1}$ with $j \in \{1,\ldots,\ell\}$. By convention, δ_0 is the identity function. We now show by induction on i that:

1) $(\delta_i(\Gamma);\delta_i(\mathcal{R}))$ is well-typed;
2) $\delta_i(\Gamma);\delta_i(\mathcal{R}) \vdash_r \delta_i(t) : \mathrm{T}$, and similarly for t';
3) $\delta_i(\Gamma);\delta_i(\mathcal{R}) \vdash_r c_j : \mathrm{Msg}$ for any $j \in \{i+1,\ldots,\ell\}$;
4) $\delta_i(t) \blacktriangleright \delta_i(t') \Rightarrow t \blacktriangleright t'$.

Base case: $i = 0$. The two first properties hold by hypothesis, whereas the last one trivially holds. It remains to establish that

$$\Gamma; \mathcal{R} \vdash_r c_j : \mathrm{Msg} \text{ for any } j \in \{1, \ldots, \ell\}.$$

Note that each c_j is a subterm in t, t', or $\mathcal{R}$. We know that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$, $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}'$, and also that $(\Gamma; \mathcal{R})$ is well-typed. By Lemma 4, we know that $\Gamma; \mathcal{R} \vdash_r c_j : \mathrm{T}_j$ for some message type T_j, and thus we have that $\Gamma; \mathcal{R} \vdash_r c_j : \mathrm{Msg}$ for any $j \in \{1, \ldots, \ell\}$.

Induction step. We assume that the properties hold for i, and we establish that they also hold for $i+1$. By induction hypothesis, we know that

$$\delta_i(\Gamma); \delta_i(\mathcal{R}) \vdash_r c_{i+1} : \mathrm{Msg},$$

and thus, as c uses a random from $\mathrm{dom}(\mathcal{R})$, we know that this typing derivation ends with an application of the rule Senc, followed potentially by subtyping steps. Thus $\delta_i(\mathcal{R})(r_{i+1}) = (p_{i+1}, k)$. We can thus apply Lemma 12 using c_{i+1}, and we deduce that the first two items hold, *i.e.*

1) $(\delta_{i+1}(\Gamma); \delta_{i+1}(\mathcal{R}))$ is well-typed; and
2) $\delta_{i+1}(\Gamma); \delta_{i+1}(\mathcal{R}) \vdash_r \delta_{i+1}(t) : \mathrm{T}$, and similarly for t'.

Regarding the third point, we use Lemma 12 again, and we rely on the fact that $\delta_{i+1}(c_j) = c_j$ for any $j \in \{i+2, \ldots, \ell\}$. They, by applying Lemma 13 to points 1 and 2 of the induction hypothesis, and using the fact that $\delta_i(\Gamma), \delta_i(\mathcal{R}) \vdash_r c_{i+1} : \mathrm{Msg}$ by induction hypothesis, we get that $\delta_{c_{i+1}} \circ \delta_i(t) \blacktriangleright \delta_{c_{i+1}} \circ \delta_i(t')$ implies $\delta_i(t) \blacktriangleright \delta_i(t')$. Therefore, using item 4 of the induction hypothesis, we deduce that:

$$\delta_{i+1}(t) \blacktriangleright \delta_{i+1}(t') \;\Rightarrow\; t \blacktriangleright t'.$$

As the ciphertexts $c_1, \ldots, c_\ell$ are numbered in such a way that when $i < j$, c_i is not a subterm of c_j (*i.e.* of p_j), composing all δ_{c_j}, we replace all ciphertexts with k, as δ_k does: *i.e.* $\delta_k(m) = (\delta_{c_\ell} \circ \cdots \circ \delta_{c_1})(m) = \delta_\ell(m)$, and this allows us to conclude. □

Definition 15. *Let $(\Gamma; \mathcal{R})$ be a well-typed mapping environment. We number $(k_i)_{1 \le i \le \ell}$ the names that are assigned a symmetric key type in Γ. We define $\delta_\Gamma = \delta_{k_\ell} \circ \ldots \circ \delta_{k_1}$, that can be applied to terms and environments.*

Note that for $k \neq k'$, δ_k and $\delta_{k'}$ commute. Thus, Definition 15 above does not depend on the choice of a particular order for keys.

Lemma 15. *Consider two ground base-logic terms t, t', a well-typed environment $(\Gamma; \mathcal{R})$ such that $(\Gamma; \mathcal{R}) \vdash_r t : \mathrm{T}$ and $(\Gamma; \mathcal{R}) \vdash_r t' : \mathrm{T}'$. Then:*

- *$(\delta_\Gamma(\Gamma), \delta_\Gamma(\mathcal{R}))$ is a well-typed environment;*
- *$(\delta_\Gamma(\Gamma), \delta_\Gamma(\mathcal{R})) \vdash_r \delta_\Gamma(t) : \mathrm{T}$;*
- *$\delta_\Gamma(t) \blacktriangleright \delta_\Gamma(t') \;\Rightarrow\; t \blacktriangleright t'$.*

Proof. Let $k_1, \ldots, k_\ell$ the names that are assigned a symmetric key type in Γ. We prove this lemma by successively applying Lemma 14 to each δ_{k_i}. □

F. Soundness of the restricted type system

In this section, we state our main result regarding the soundness of the restricted type system, *i.e.* that when a ground base-logic term t has type T in the restricted type system, then t is in the interpretation of T.

Theorem 2. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, t a ground base-logic term, and T a message type. If $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$, then $\Gamma; \mathcal{R} \models t : \mathrm{T}$.*

Proof. Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, and t, T such that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$. We must prove that $\Gamma; \mathcal{R} \models t : \mathrm{T}$, *i.e.*, according to Definition 5, that there exists t' such that $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}$ and $t' \approx t$, plus an additional requirement which depends on T. We prove that by induction on the typing derivation Π for the judgement $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$.

We choose $t' = t$, so that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$, by hypothesis, and $t \approx t$. For the last requirement, we examine each possible case regarding the last rule of Π.

- Rules Zeros, Fun-Low, Fun-Msg, Senc, as well as rule Name when $\mathrm{T} = \mathrm{Low}$. In such a case, there is nothing to prove since interpretations of type Low and Msg do not have an additional requirement.
- Rules Cst-0 and Cst-∞. In such a case, t is a constant c, and therefore we have $t \approx \mathsf{c}$ which is exactly the condition we had to establish.
- Rule Pair. In such a case, we have $t = \langle t_1, t_2 \rangle$, $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$, and $\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{T}_i$ for $i = 1, 2$ (from the rule's premises). Applying our induction hypothesis, we deduce that $\Gamma; \mathcal{R} \models t_i : \mathrm{T}_i$, with $i = 1, 2$. To conclude, it remains to use Lemma 9 with $\mathsf{fst}(t) \approx t_1$ and $\mathsf{snd}(t) \approx t_2$.
- Rules Eq, Eq-True-Cst, Eq-False-Cst, and Eq-False. In such a case, we have $t = (t_1 = t_2)$. In the case of rule Eq, $\mathrm{T} = \mathrm{Bool}$, and we must show that $(t_1 = t_2) \mathrel{\tilde{\in}} \{\mathsf{true}, \mathsf{false}\}$. The proof is immediate, as by assumption $[\![=]\!]^{\mathbb{M}}$ can only produce $[\![\mathsf{true}]\!]^{\mathbb{M}}$ or $[\![\mathsf{false}]\!]^{\mathbb{M}}$ for any $\mathbb{M}$.
  In the case of rule Eq-True-Cst, $\mathrm{T} = \mathrm{Cst}^0_{\mathsf{true}}$ and we must show that $(t_1 = t_2) \approx \mathsf{true}$. Thanks to the induction hypothesis, we have $\Gamma; \mathcal{R} \models t_i : \mathrm{Cst}^0_{\mathsf{c}}$ for $i = 1, 2$ and for some c. This means that $t_i \approx \mathsf{c}$, and thus $t_1 \approx t_2$, which implies that $(t_1 = t_2) \approx \mathsf{true}$.
  In the case of rule Eq-False-Cst, with the same reasoning, we get $\Gamma; \mathcal{R} \models t_i : \mathrm{Cst}^\star_{\mathsf{c}_i}$ with $\mathsf{c}_1 \neq \mathsf{c}_2$. The cases where $\star$ is 0 and ∞ are similar. We only detail the case where $\Gamma; \mathcal{R} \models t_1 : \mathrm{Cst}^\infty_{\mathsf{c}_1}$ and $\Gamma; \mathcal{R} \models t_2 : \mathrm{Cst}^0_{\mathsf{c}_2}$. By definition, this means $t_1 \mathrel{\tilde{\in}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}_1}$ and $t_2 \approx \mathsf{c}_2$. Since set $\mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}_1}$ does not contain c_2 (by construction), and since distinct constants are interpreted as different values, t_1 and t_2 are interpreted as different values overwhelmingly, *i.e.* $(t_1 = t_2) \approx \mathsf{false}$. Therefore $\Gamma; \mathcal{R} \models t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}$.
  Finally, in the case of rule Eq-False, we have $\Gamma; \mathcal{R} \vdash_r t_1 : \mathrm{High}$ and $\Gamma; \mathcal{R} \vdash_r t_2 : \mathrm{Low}$. Applying the induction hypothesis, we obtain $\Gamma; \mathcal{R} \models t_1 : \mathrm{High}$ and $\Gamma; \mathcal{R} \models t_2 : \mathrm{Low}$. Therefore, by definition of the interpretation of High, we know that $t_2 \blacktriangleright t_1$, *i.e.* an attacker cannot use t_2 to produce a result equals to t_1 with a non-negligible probability of success, in any model. In particular, with

an attacker that simply returns its input, this means that for any computational model $\mathbb{M}$:

$$\mathbb{P}_\rho\left[[\![t_2]\!]^{\mathbb{M}}(1^\eta,\rho)) = [\![t_1]\!]^{\mathbb{M}}(1^\eta,\rho)\right] \in \mathsf{negl}(\eta).$$

Hence $(t_1 = t_2) \approx \mathsf{false}$, and $\Gamma;\mathcal{R} \models t_1 = t_2 : \mathrm{Cst}^0_{\mathsf{false}}$.

- Rule Sub-Typing. In such a case we have $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T}'$ for some T' such that $\mathrm{T}' \leq \mathrm{T}$. Applying the induction hypothesis, we deduce that $\Gamma;\mathcal{R} \models t : \mathrm{T}'$, and thanks to Lemma 10, we conclude that $\Gamma;\mathcal{R} \models t : \mathrm{T}$.
- Rule If. In such a case, we have:
  - $t = \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$,
  - $\Gamma;\mathcal{R} \vdash_r t_0 : \mathrm{Bool}$, and
  - $\Gamma;\mathcal{R} \vdash_r t_i : \mathrm{T}$ for $i = 1, 2$.

  By the induction hypothesis, we have $\Gamma;\mathcal{R} \models t_0 : \mathrm{Bool}$, and $\Gamma;\mathcal{R} \models t_i : \mathrm{T}$ for $i = 1, 2$.
  In addition, we can show that $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T} + \mathrm{T}$, by applying rule Sub-Typing to $\Gamma;\mathcal{R} \vdash_r t : \mathrm{T}$. Therefore, by definition, $\Gamma;\mathcal{R} \models t : \mathrm{T} + \mathrm{T}$. Then, we conclude that $\Gamma;\mathcal{R} \models t : \mathrm{T}$ thanks to Lemma 11.
- Rule Name when $\mathrm{T} = \mathrm{High}$. In such a case, t is some name n such that $\Gamma(n) = \mathrm{High}$, and we have to establish that $\Gamma;\mathcal{R} \models n : \mathrm{High}$. Let t_L be a ground term such that $\Gamma;\mathcal{R} \models t_L : \mathrm{Low}$. We have to establish that $t_L \blacktriangleright n$.
  By definition of $\models$, we know that there exists $t'_L \approx t_L$ such that $\Gamma;\mathcal{R} \vdash_r t'_L : \mathrm{Low}$. In order to conclude, we would like to use Lemma 15 on $\Gamma;\mathcal{R} \vdash_r t'_L : \mathrm{Msg}$ and $\Gamma;\mathcal{R} \vdash_r n : \mathrm{High}$. Its hypotheses are satisfied, except potentially the requirement that $(\Gamma;\mathcal{R})$ is well-typed: we only know it to be well-formed. Let $\mathcal{R}'$ be the mapping such that $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}) \cap \{r \mid r \text{ occurs in } t'_L\}$. Thanks to Lemma 2, we know that $\Gamma;\mathcal{R}' \vdash_r t'_L : \mathrm{Low}$. Moreover, for any $r \in \mathrm{dom}(\mathcal{R}')$, we know that $\mathsf{senc}(p,k,r)$ occurs in t'_L for some p and some k. Relying on Lemma 4, we can deduce that $\Gamma;\mathcal{R}' \vdash_r \mathsf{senc}(p,k,r) : \mathrm{Msg}$, and since $r \in \mathrm{dom}(\mathcal{R}')$, we deduce that this derivation ends with an instance of the rule Senc (potentially followed by subtyping steps), and thus $\mathcal{R}'(r) = (p,k)$. Therefore, $(\Gamma;\mathcal{R}')$ is well-typed.
  Applying Lemma 15 to $\Gamma;\mathcal{R}' \vdash_r t'_L : \mathrm{Low}$ and $\Gamma;\mathcal{R}' \vdash_r n : \mathrm{High}$, we obtain that $\delta_\Gamma(\Gamma);\delta_\Gamma(\mathcal{R}') \vdash_r \delta_\Gamma(t'_L) : \mathrm{Low}$ and that it suffices to prove $\delta_\Gamma(t'_L) \blacktriangleright n$ to deduce that $t'_L \blacktriangleright n$, and thus $t_L \blacktriangleright n$ (since $\delta_\Gamma(n) = n$ and $t'_L \approx t_L$).
  Before doing so, we characterise the positions in $\delta_\Gamma(t'_L)$ where n can occur. By Lemma 6, each occurrence of n in $\delta_\Gamma(t'_L)$ is in a context of the form $C_1[\mathsf{senc}(C_2,k,r)]$ with a key in $\delta_\Gamma(\Gamma)$ and $r \in \mathrm{dom}(\mathcal{R}')$, $C_1[\mathsf{zeros}(C_2)]$, $C_1[C_2 = t']$, or $C_1[t' = C_2]$. By construction, all encryptions in $\delta_\Gamma(t'_L)$ with a key in Γ and a random symbol in $\mathrm{dom}(\mathcal{R}')$ are of the form $\mathsf{senc}(\mathsf{zeros}(t'),k,r)$ for some t'. Thus, the first case is absorbed by the second, *i.e.* the context contains zeros or $=$ above n. In other words, if t_i for $1 \leq i \leq \ell$ are all the subterms of t of the form $\mathsf{zeros}(t')$ or $t' = t''$, each occurence of n in t appears inside a term t_i.
  We can now prove that $\delta_\Gamma(t'_L) \blacktriangleright n$. Fix a computational model $\mathbb{M}$, and a PPTM adversary $\mathcal{A}$, with access to the adversarial part ρ_a of the random tape, and $[\![\delta_\Gamma(t'_L)]\!]^{\mathbb{M}}(1^\eta,\rho)$. We must show that the probability (in ρ) that $\mathcal{A}$ computes $[\![n]\!]^{\mathbb{M}}(1^\eta,\rho)$ is negligible. We construct a PPTM adversary $\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b})$, with access to a tape $\rho' = (\rho_\mathsf{h} \setminus \{n\}, \rho_\mathsf{a})$, where $\rho_\mathsf{h} \setminus \{n\}$ denotes the random tape ρ_h where the part $\rho_\mathsf{h}(n)$ associated to n was removed, and ρ_b, an additional source of randomness unused by $\mathcal{A}$. The machine $\mathcal{B}$ internally runs $\mathcal{A}$, using the adversarial part of ρ' to simulate ρ_a for $\mathcal{A}$. It returns the value produced by $\mathcal{A}$.
  To provide $\mathcal{A}$ with $[\![\delta_\Gamma(t'_L)]\!]^{\mathbb{M}}(1^\eta,\rho)$, $\mathcal{B}$ needs to compute this value. To do so, $\mathcal{B}$ goes through the term $\delta_\Gamma(t'_L)$ recursively.
  - When encountering a name other than n, $\mathcal{B}$ obtains its value from ρ' (it fails when encountering n).
  - When encountering $\mathsf{f}(t'_1,\ldots,t'_k)$ for some function symbol f other than zeros or $=$, $\mathcal{B}$ recursively computes the interpretation of each t'_i, and calls $[\![\mathsf{f}]\!]^{\mathbb{M}}$.
  - To evaluate $\mathsf{zeros}(t')$ or $t' = t''$ when t' and t'' do not contain n, $\mathcal{B}$ proceeds as in the previous case.
  - To evaluate $t' = t''$ otherwise: $\mathcal{B}$ draws a random bit from ρ_b.
  - Similarly, to evaluate $\mathsf{zeros}(t')$ if n occurs in t': since all functions are interpreted as polynomial time algorithms, there exists a polynomial $p(\eta)$ that bounds the length of $[\![t']\!]^{\mathbb{M}}(1^\eta,\rho)$ for all subterms t' of $\delta_\Gamma(t'_L)$, all η and ρ. $\mathcal{B}$ draws uniformly at random (from ρ_b) an integer l between 0 and $p(\eta)$, and return a string of l zeros.

  By the observation on the positions of n in $\delta_\Gamma(t'_L)$, $\mathcal{B}$ never needs to evaluate n. Moreover, the number g of guesses made by $\mathcal{B}$ is fixed and independent of η, ρ, ρ_b: it only depends on the structure of term t'_L. The probability (in ρ_b) of success of each independent guess, for any η, ρ, is at least $p(\eta)^{-1}$ (assuming w.l.o.g. that $p(\eta) \geq 2$, for the case of booleans). Thus, if we let $q(\eta) = p(\eta)^g$, for any η, ρ, the probability in ρ_b of $\mathcal{B}$ correctly guessing each time is at least $q(\eta)^{-1}$.
  Hence, for any η,ρ, if $\Delta_{\eta,\rho}$ denotes $[\![\delta_\Gamma(t'_L)]\!]^{\mathbb{M}}(1^\eta,\rho)$, we have $\mathbb{P}_{\rho_\mathsf{b}}[\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b}) = \mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho})] \geq q(\eta)^{-1}$. This bound is uniform in ρ, and therefore carries over when taking the probability over ρ. More precisely, for any η, the conditional probability on ρ of that equality, restricted to the ρ for which $\mathcal{A}$ succeeds in computing n, satisfies

$$\mathbb{P}_{\rho,\rho_\mathsf{b}}[\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b}) = \mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) \mid \mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\Big] \geq q(\eta)^{-1}$$

  and is therefore non-negligible. Moreover, since $\rho(n)$ is drawn independently from all parameters of $\mathcal{B}$, $\mathbb{P}_{\rho,\rho_\mathsf{b}}\left[\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\right]$ is negligible.
  On the other hand, for any η:

$$\mathbb{P}_{\rho,\rho_\mathsf{b}}\left[\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\right] \geq \mathbb{P}_{\rho,\rho_\mathsf{b}}[\mathcal{B}(1^\eta,\rho',\rho_\mathsf{b}) = \mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) \mid \mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\Big] \times \mathbb{P}_{\rho,\rho_\mathsf{b}}\left[\mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\right].$$

Therefore, $\mathbb{P}_{\rho,\rho_\mathsf{b}}\left[\mathcal{A}(1^\eta,\rho_\mathsf{a},\Delta_{\eta,\rho}) = [\![n]\!]^{\mathbb{M}}(1^\eta,\rho)\right]$ is negligible, which concludes the proof.

□

APPENDIX D
TYPE SYSTEM FOR MESSAGES (BASE-LEVEL)

The typing system we consider in this section is composed of rules in Figure 1 and Figure 2, except that it manipulates terms of the base logic, and thus non-indexed symbols. In rules NAME, SDEC, SENC, CST-∞, the indices are removed: n, k, r and c are non-indexed symbols (from the base logic).

Definition 16. *The base logic type system is composed of the rules given in Figure 1 and Figure 2 where $\vdash$ is replaced with $\vdash_b$, and that manipulates base logic terms (i.e. non indexed symbols).*

A typing judgement w.r.t. this typing system is an expression of the form $\Gamma; R \vdash_b t : \mathrm{T}$ where $(\Gamma; R)$ is a well-formed typing environment, t is a base logic term (that may contain variables), and T is a message type.

We aim at establishing the following result, i.e. the soundness of the type system introduced in this section.

Theorem 3. *Let $(\Gamma; R)$ be a well-formed typing environment, t a ground term of the base logic, and T a message type. Then $\Gamma; R \vdash_b t : \mathrm{T}$ implies $\Gamma; \mathcal{R} \models t : \mathrm{T}$ for some $\mathcal{R}$.*

A. Preliminaries

A *substitution* θ is a mapping from variables to ground base logic terms. We write $\mathrm{dom}(\theta)$ its *domain*, and $t\theta$ the application of θ on a term t. We say that θ is grounding for t when $\mathrm{vars}(t) \subseteq \mathrm{dom}(\theta)$, *i.e.* $t\theta$ is ground.

Definition 17. *Let $(\Gamma; \mathcal{R})$ be a well-formed environment, and θ be a substitution. We say that θ is* well-typed *w.r.t. $(\Gamma; \mathcal{R})$, denoted by $\Gamma; \mathcal{R} \vdash_r \theta$, if $\mathrm{dom}(\theta) \subseteq \mathrm{vars}(\Gamma)$, and*

$$\Gamma; \mathcal{R} \vdash_r \theta(x) : \Gamma(x) \text{ for any } x \in \mathrm{dom}(\theta).$$

B. Dealing with cryptography

Lemma 16. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, T be a message type, $\Gamma(k) = \mathrm{SK[T]}$, and t be a ground term such that $\Gamma; \mathcal{R} \vdash_r t : \mathrm{Msg}$. Let $\mathsf{senc}(p_1, k, r_1), \ldots, \mathsf{senc}(p_\ell, k, r_\ell)$ be the sequence of encryptions with key k and random symbols from $\mathrm{dom}(\mathcal{R})$ occurring in t, and*

$$\delta(t,k) = \begin{cases} \text{if } t = \mathsf{senc}(p_1,k,r_1) \text{ then } p_1 \text{ else} \\ \quad \ldots \\ \quad \text{if } t = \mathsf{senc}(p_\ell,k,r_\ell) \text{ then } p_\ell \text{ else fail} \end{cases}$$

We have that $\mathsf{sdec}(t,k) \approx \delta(t,k)$.

Proof. Consider a computational model $\mathbb{M}$, and fix some η, and some ρ.

If there exists $i \in \{1,\ldots,\ell\}$ such that $[\![t]\!]^{\mathbb{M}}(\eta,\rho) = [\![\mathsf{senc}(p_i,k,r_i)]\!]^{\mathbb{M}}(\eta,\rho)$, then consider a smallest such i: by correctness of the encryption scheme, we have that $[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![p_i]\!]^{\mathbb{M}}(\eta,\rho)$ for that i. Thus, $[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![\delta(t,k)]\!]^{\mathbb{M}}(\eta,\rho)$.

On the other hand, if for all $i \in \{1,\ldots,\ell\}$, we have that $[\![t]\!]^{\mathbb{M}}(\eta,\rho) \neq [\![\mathsf{senc}(p_i,k,r_i)]\!]^{\mathbb{M}}(\eta,\rho)$, and $[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![\mathsf{fail}]\!]^{\mathbb{M}}(\eta,\rho)$, then we also have $[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![\delta(t,k)]\!]^{\mathbb{M}}(\eta,\rho)$. These two observations establish that for all η,

$$\begin{aligned} \mathbb{P}_\rho\Big[\exists i.\ [\![t]\!]^{\mathbb{M}}(\eta,\rho) = [\![\mathsf{senc}(p_i,k,r_i)]\!]^{\mathbb{M}}(\eta,\rho) \\ \vee\ [\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![\mathsf{fail}]\!]^{\mathbb{M}}(\eta,\rho)\Big] \\ \leq \mathbb{P}_\rho\Big[[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) = [\![\delta(t,k)]\!]^{\mathbb{M}}(\eta,\rho)\Big] \end{aligned}$$

It is therefore sufficient to show that the probability of $\forall i.\ [\![t]\!]^{\mathbb{M}}(\eta,\rho) \neq [\![\mathsf{senc}(p_i,k,r_i)]\!]^{\mathbb{M}}(\eta,\rho) \wedge [\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) \neq [\![\mathsf{fail}]\!]^{\mathbb{M}}(\eta,\rho)$ is negligible. Assuming it is not, we construct an adversary that breaks the INT-CTXT assumption (see Appendix).

By Lemma 8, since $\Gamma; \mathcal{R} \vdash_r t : \mathrm{Msg}$, t is evaluable in $(\Gamma; \mathcal{R})$, *i.e.* $\mathbb{P}_\rho\left[\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_\mathsf{a}) = [\![t]\!]^{\mathbb{M}}(1^\eta,\rho_\mathsf{a})\right] \in \mathsf{ow}(\eta)$.

We construct an adversary $\mathcal{A}(1^\eta,\rho')$ against the INT-CTXT game, where ρ' is w.l.o.g. seen as $(\rho'_\mathsf{h},\rho_\mathsf{a})$, ρ'_h being the part of the random tape ρ_h not associated with the symbol k or any encryption randomness symbol r associated with k in Γ. $\mathcal{A}$ internally simulates the execution of $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_\mathsf{a})$, and its definition relies on Γ and $\mathcal{R}$. He answers the queries to oracle $\mathcal{O}$ as follows.

- $\mathcal{O}(\text{“}n\text{”})$: if “$n$” is the symbol of a key in Γ, then fail. Otherwise read its associated value $[\![n]\!]^{\mathbb{M}}(1^\eta,\rho)$ on ρ'_h, and return it.
- $\mathcal{O}(m, \text{“}k_1\text{”}, \text{“}r\text{”})$: if $k_1 = k$, call his own encryption oracle to answer the call. Otherwise, check if r is associated with k_1 in $\mathcal{R}$. If so, obtain the values associated with r and k_1 on ρ'_h, then apply $[\![\mathsf{senc}]\!]^{\mathbb{M}}$ to m, and fail otherwise.

Once $\mathcal{A}$ obtains the value returned by $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_\mathsf{a})$, he submits it as his INT-CTXT challenge.

Whenever $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)$ successfully evaluates, $\mathcal{A}$ accurately simulates the oracle $\mathcal{O}$ (up to bijection between the part of ρ_h for k and its randoms, and the tape used by the INT-CTXT encryption oracle). Thus the execution of $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)$ is accurately simulated, and $\mathcal{A}$ obtains and submits the value $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta,\rho_\mathsf{a})$. With overwhelming probability, this value is equal to $[\![t]\!]^{\mathbb{M}}(1^\eta,\rho_\mathsf{a})$. Moreover, by assumption, with non-negligible probability, $\forall i.[\![t]\!]^{\mathbb{M}}(1^\eta,\rho_\mathsf{a}) \neq [\![\mathsf{senc}(p_i,k,r_i)]\!]^{\mathbb{M}}(\eta,\rho)$, and $[\![\mathsf{sdec}(t,k)]\!]^{\mathbb{M}}(\eta,\rho) \neq [\![\mathsf{fail}]\!]^{\mathbb{M}}(\eta,\rho)$, thus with non-negligible probability, both events occur, *i.e.* $\mathcal{A}$ submits a value that lets him win the INT-CTXT game. This concludes the proof. □

C. Main result of this section

In order to establish the sounndness of our type system, we will prove a lemma, which states that a well-typed term t can be transformed into another equivalent term t' that can be typed in the restricted system. Then, we will conclude relying on the soundness result established for the restricted system. In order to be able to establish such a lemma by induction and to deal with the rule ASSIGN, we have to show the result for any non ground term that is closed by a well-typed substitution. The lemma is as follows:

Lemma 17. *Let t be a term of the base logic, $(\Gamma; R)$ be a well-formed typing environment, $(\Gamma; \mathcal{R})$ be a mapping typing environment that is also well-formed such that $\mathrm{dom}(\mathcal{R}) \cap R = \emptyset$, and θ be a substitution grounding for t. If $\Gamma; R \vdash_b t : \mathrm{T}$ and $\Gamma; \mathcal{R} \vdash_r \theta$, then there exist a ground term t', and a mapping $\mathcal{R}'$ such that:*

1) $dom(\mathcal{R}') \subseteq R$;
2) $t\theta \approx t'$; and
3) $\Gamma, \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}$.

Note that, once this lemma will be established, Theorem 3 will follow.

Proof of Theorem 3. Let $(\Gamma; R)$ be a well-formed typing environment, t a ground term of the base logic, and T a message type such that $\Gamma; R \vdash_b t : \mathrm{T}$. Applying Lemma 17 with the empty substitution, we deduce that there exist t' and $\mathcal{R}'$ such that $t \approx t'$ and $\Gamma; \mathcal{R} \vdash_r t' : \mathrm{T}$, and thus $\Gamma; \mathcal{R}' \models t' : \mathrm{T}$ thanks to Theorem 2, and this allows us to deduce that $\Gamma; \mathcal{R} \models t : \mathrm{T}$ relying on Lemma 9. □

It remains to establish Lemma 17.

Proof. We establish the lemma by induction on the typing derivation witnessing the fact that $\Gamma; R \vdash_b t : \mathrm{T}$. When the rule ending the typing derivation exists in the restricted type system, the result is a direct consequence of the application of the induction hypothesis. Thus, in addition to the rules VAR, ASSIGN, BREAK-SUM, FST, SND, IF-TRUE, IF-FALSE, and SDEC that have no counterpart in the restricted type system, we detail the rule PAIR below to illustrate an easy case of the proof. We also detail the rule SENC to illustrate the construction of $\mathcal{R}'$. Note that for each axiom, we set $\mathcal{R}'$ to the empty mapping.

Case of the rule PAIR:

$$\frac{\Gamma; R_1 \vdash_b t_1 : \mathrm{T}_1 \quad \Gamma; R_2 \vdash_b t_2 : \mathrm{T}_2}{\Gamma; R_1 \sqcup R_2 \vdash_b \langle t_1, t_2 \rangle : \mathrm{T}_1 \times \mathrm{T}_2}$$

We have that $R = R_1 \sqcup R_2$, $t = \langle t_1, t_2 \rangle$, and $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$. We have that $(\Gamma; R_i)$ is well-formed and $\mathrm{dom}(\mathcal{R}) \cap R_i = \emptyset$ for $i = 1, 2$, therefore applying our induction hypothesis with θ on both subtrees, we deduce that there exist ground t'_i and $\mathcal{R}'_i$ (for $i = 1, 2$) such that:

1) $\mathrm{dom}(\mathcal{R}'_i) \subseteq R_i$;
2) $t_i\theta \approx t'_i$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_i \vdash_r t'_i : \mathrm{T}_i$.

Let $t' = \langle t'_1, t'_2 \rangle$, and $\mathcal{R}' = \mathcal{R}'_1 \sqcup \mathcal{R}'_2$. We have that:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_1) \sqcup \mathrm{dom}(\mathcal{R}'_2) \subseteq R_1 \sqcup R_2 = R$;
2) $t\theta = \langle t_1\theta, t_2\theta \rangle \approx \langle t'_1, t'_2 \rangle = t'$; and
3) We have seen that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_i \vdash_r t'_i : \mathrm{T}_i$ for $i = 1, 2$, and therefore we have that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_1 \sqcup \mathcal{R}'_2 \vdash_r t'_i : \mathrm{T}_i$ for $i = 1, 2$ (thanks to Lemma 3). Applying the rule PAIR of the restricted type system, we deduce that $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}$.

Case of the rule SENC:

$$\frac{\Gamma; R \vdash_b t : \mathrm{T} \quad k : \mathrm{SK[T]} \in \Gamma}{\Gamma; R \sqcup \{r\} \vdash_b \mathsf{senc}(t, k, r) : \mathrm{Low}}$$

By induction hypothesis, there exist t'_0 and $\mathcal{R}'_0$ such that:

1) $\mathrm{dom}(\mathcal{R}'_0) \subseteq R$;
2) $t\theta \approx t'_0$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \vdash_r t'_0 : \mathrm{T}$.

Let $t' = \mathsf{senc}(t'_0, k, r)$ and $\mathcal{R}' := \mathcal{R}'_0 \sqcup \{r \mapsto (t'_0, k)\}$. We then have:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_0) \sqcup \{r\} \subseteq R \sqcup \{r\}$;
2) $\mathsf{senc}(t, k, r)\theta = \mathsf{senc}(t\theta, k, r) \approx \mathsf{senc}(t'_0, k, r) = t'$; and
3) Since $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \vdash_r t'_0 : \mathrm{T}$, thanks to Lemma 3, we have that $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t'_0 : \mathrm{T}$ We also have $\Gamma(k) = \mathrm{SK[T]}$, and $\mathcal{R}'(r) = (t'_0, k)$. We can then apply the rule SENC of the restricted system to get $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{Low}$.

Case of the rule VAR:

$$\frac{\Gamma(x) = \mathrm{T}}{\Gamma; R \vdash_b x : \mathrm{T}}$$

Let $t' = x\theta$ and $\mathcal{R}'$ the empty mapping. We have that:

1) $\mathrm{dom}(\mathcal{R}') = \emptyset \subseteq R$;
2) $t' \approx x\theta$; and
3) Since $\Gamma; \mathcal{R} \vdash_r \theta$, we have that $\Gamma; \mathcal{R} \vdash_r \theta(x) : \Gamma(x)$, and therefore we have that $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}$.

Case of the rule ASSIGN:

$$\frac{\Gamma, x : \mathrm{T}'; R_1 \vdash_b t_1 : \mathrm{T} \quad \Gamma; R_2 \vdash_b t_2 : \mathrm{T}'}{\Gamma; R_1 \sqcup R_2 \vdash_b t_1[x \mapsto t_2] : \mathrm{T}}$$

We have that $R = R_1 \sqcup R_2$, and $t = t_1[x \mapsto t_2]$. We have that $(\Gamma; R_2)$ is well-formed, and $\mathrm{dom}(\mathcal{R}) \cap R_2 = \emptyset$, therefore applying our induction hypothesis on the second subtree, we deduce that there exist t'_2, and $\mathcal{R}'_2$ such that:

1) $\mathrm{dom}(\mathcal{R}'_2) \subseteq R_2$;
2) $t_2\theta \approx t'_2$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_2 \vdash_r t'_2 : \mathrm{T}'$.

Let $\theta_1 = \theta \cup \{x \mapsto t'_2\}$. In order to apply our induction hypothesis on the first subtree using θ_1, we first establish that $\Gamma, x : \mathrm{T}'; \mathcal{R} \sqcup \mathcal{R}'_2 \vdash_r \theta_1$.

- Since $\Gamma; \mathcal{R} \vdash_r \theta$, we have $\Gamma; \mathcal{R} \vdash_r \theta(y) : \Gamma(y)$ for each $y \in \mathrm{dom}(\theta)$, and thus for each each $y \in \mathrm{dom}(\theta_1) \setminus \{x\}$, thanks to Lemma 3, we have that

$$\Gamma, x : \mathrm{T}'; \mathcal{R} \cup \mathcal{R}'_2 \vdash_r \theta_1(y) : \Gamma(y).$$

- $\Gamma, x : \mathrm{T}'; \mathcal{R} \cup \mathcal{R}'_2 \vdash_r \theta_1(x) : \Gamma(x)$ as we have that $\theta_1(x) = t'_2$ and $\Gamma(x) = \mathrm{T}'$ (thanks to Lemma 3).

As θ is grounding for $t_1[x \mapsto t_2]$, we have that θ_1 is grounding for t_1. We have also that $\mathrm{dom}(\mathcal{R} \sqcup \mathcal{R}'_2) \cap R_1 = \emptyset$, and thus, applying our induction hypothesis on the first subtree using the substitution θ_1 for which we have that $\Gamma, x : \mathrm{T}'; \mathcal{R} \sqcup \mathcal{R}'_2 \vdash_r \theta_1$, we deduce that there exist t'_1, and $\mathcal{R}'_1$ such that:

1) $\mathrm{dom}(\mathcal{R}'_1) \subseteq R_1$;
2) $t_1\theta_1 \approx t'_1$; and
3) $\Gamma, x : \mathrm{T}'; \mathcal{R} \sqcup \mathcal{R}'_2 \sqcup \mathcal{R}'_1 \vdash_r t'_1 : \mathrm{T}$.

Let $t' = t'_1$, and $\mathcal{R}' = \mathcal{R}'_1 \sqcup \mathcal{R}'_2$. We will show that the requirements are satisfied:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_1) \sqcup \mathrm{dom}(\mathcal{R}'_2) \subseteq R_1 \sqcup R_2 = R$.
2) $t\theta = (t_1[x \mapsto t_2])\theta = t_1\theta[x \mapsto t_2\theta] \approx t_1\theta_1 \approx t'_1 = t'$.
3) Relying on Lemma 2 and the fact that $t' = t'_1$, and $\mathcal{R}' = \mathcal{R}'_1 \sqcup \mathcal{R}'_2$, we deduce that $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}$.

Case of the rule FST:

$$\frac{\Gamma; R \vdash_b t_0 : \mathrm{T}_1 \times \mathrm{T}_2}{\Gamma; R \vdash_b \mathsf{fst}(t_0) : \mathrm{T}_1}$$

We have that $t = \mathsf{fst}(t_0)$, and $\mathrm{T} = \mathrm{T}_1$. Applying our induction hypothesis on the subtree, we deduce that there exist t'_0 and $\mathcal{R}'_0$ such that:

1) $\mathrm{dom}(\mathcal{R}'_0) \subseteq R$;
2) $t_0\theta \approx t'_0$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \vdash_r t'_0 : \mathrm{T}_1 \times \mathrm{T}_2$.

Thanks to Theorem 2, we have $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \models t'_0 : \mathrm{T}_1 \times \mathrm{T}_2$ which means in particular that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \models \mathsf{fst}(t'_0) : \mathrm{T}_1$. By definition of type interpretation, we have a term t' such that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \vdash_r t' : \mathrm{T}_1$ and $t' \approx \mathsf{fst}(t'_0)$. Let $\mathcal{R}' = \mathcal{R}'_0$. We show that the requirements are satisfied:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_0) \subseteq R$;
2) $t' \approx \mathsf{fst}(t'_0) \approx \mathsf{fst}(t_0\theta) = (\mathsf{fst}(t_0))\theta = t\theta$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}_1$.

The case of the rule SND can be done in a similar way.

Case of the rule IF-TRUE:

$$\frac{\Gamma; R_0 \vdash_b t_0 : \mathrm{Cst}^0_{\mathsf{true}} \quad \Gamma; R_1 \vdash_b t_1 : \mathrm{T}_1}{\Gamma; R_0 \sqcup R_1 \vdash_b \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : \mathrm{T}_1}$$

In such a case, we have that $t = \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$, $R = R_0 \sqcup R_1$, and $\mathrm{T} = \mathrm{T}_1$. We have that $(\Gamma; R_i)$ is well-formed and $\mathrm{dom}(\mathcal{R}) \cap R_i = \emptyset$ for $i = 0, 1$, therefore applying our induction hypothesis on both subtrees, we deduce that there exist t'_i and $\mathcal{R}'_i$ (for $i = 0, 1$) such that:

1) $\mathrm{dom}(\mathcal{R}'_i) \subseteq R_i$;
2) $t_i\theta \approx t'_i$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_i \vdash_r t'_i : \mathrm{T}_i$ with $\mathrm{T}_0 = \mathrm{Cst}^0_{\mathsf{true}}$.

Let $t' = t'_1$, and $\mathcal{R}' = \mathcal{R}'_1$. We will show that the requirements are satisfied:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_1) \subseteq R_1$;
2) We have seen that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \vdash_r t'_0 : \mathrm{Cst}^0_{\mathsf{true}}$. Thanks to Theorem 2, we deduce that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_0 \models t'_0 : \mathrm{Cst}^0_{\mathsf{true}}$ which means that $t'_0 \approx \mathsf{true}$. Therefore, we have that:

$$\begin{array}{rcl} t\theta & = & \mathsf{if}\ t_0\theta\ \mathsf{then}\ t_1\theta\ \mathsf{else}\ t_2\theta \\ & \approx & \mathsf{if}\ t'_0\ \mathsf{then}\ t'_1\ \mathsf{else}\ t_2\theta \\ & \approx & \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ t'\ \mathsf{else}\ t_2\theta \\ & \approx & t'. \end{array}$$

3) We have seen that $\Gamma; \mathcal{R} \sqcup \mathcal{R}'_1 \vdash_r t'_1 : \mathrm{T}_1$.

The case of the rule IF-FALSE can be done in a similar way.

Case of the rule BREAK-SUM:

$$\frac{\Gamma, x : \mathrm{T}_1; R_1 \vdash_b t : \mathrm{T} \quad \Gamma, x : \mathrm{T}_2; R_2 \vdash_b t : \mathrm{T}}{\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; R_1 \sqcup R_2 \vdash_b t : \mathrm{T}}$$

We have that $R = R_1 \sqcup R_2$. By hypothesis, we know that $\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \vdash_r \theta$, so we have that

$$\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \vdash_r \theta(x) : \mathrm{T}_1 + \mathrm{T}_2.$$

Let Γ' be the environment Γ without the mapping on variables. Thanks to Lemma 2, we have that $\Gamma'; \mathcal{R} \models \theta(x) : \mathrm{T}_1 + \mathrm{T}_2$.

By Theorem 2, we deduce that $\Gamma'; \mathcal{R} \models \theta(x) : \mathrm{T}_1 + \mathrm{T}_2$. By definition, this means that there exists a ground term $m \approx \theta(x)$ such that $\Gamma'; \mathcal{R} \vdash_r m : \mathrm{T}$ and we are in one of the three following cases:

a) $\Gamma'; \mathcal{R} \models m : \mathrm{T}_1$; or
b) $\Gamma'; \mathcal{R} \models m : \mathrm{T}_2$; or
c) there are ground terms b, m_1, and m_2 with $m = \mathsf{if}\ b\ \mathsf{then}\ m_1\ \mathsf{else}\ m_2$ and $\Gamma'; \mathcal{R} \models b : \mathrm{Bool}$, and $\Gamma'; \mathcal{R} \models m_i : \mathrm{T}_i$ for $i = 1, 2$.

Regarding case a) (and the proof is similar for case b)), let $\overline{\theta}$ be a substitution such that $\mathrm{dom}(\overline{\theta}) = \mathrm{dom}(\theta) \setminus \{x\}$, and $\overline{\theta} = \theta|_{\mathrm{dom}(\overline{\theta})}$. We have $\Gamma'; \mathcal{R} \models m : \mathrm{T}_1$, so there is m' such that $m' \approx m$, and $\Gamma'; \mathcal{R} \vdash_r m' : \mathrm{T}_1$. Γ' does not contain variables, so m' is closed.

Let $\theta' = \overline{\theta} \cup \{x \mapsto m'\}$. Before applying our induction hypothesis on the first subtree using the substitution θ', we need to show that $\Gamma, x : \mathrm{T}_1; \mathcal{R} \vdash_r m' : \mathrm{T}_1$, and this is actually an easy consequence of the fact that $\Gamma'; \mathcal{R} \vdash_r m' : \mathrm{T}_1$ by enriching of the environment with Lemma 3. Therefore, thanks to our induction hypothesis, we deduce that there exists a ground term t'_1 and a mapping $\mathcal{R}'_1$ such that:

1) $\mathrm{dom}(\mathcal{R}'_1) \subseteq R_1$;
2) $t\theta' \approx t'_1$; and
3) $\Gamma, x : \mathrm{T}_1; \mathcal{R} \sqcup \mathcal{R}'_1 \vdash_r t'_1 : \mathrm{T}$.

Let $t' = t'_1$ and $\mathcal{R}' = \mathcal{R}'_1$. We show that the requirements are satisfied:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_1) \subseteq R_1 \subseteq R_1 \sqcup R_2 = R$;
2) $t' = t'_1 \approx t\theta' = t(\overline{\theta} \cup \{x \mapsto m'\}) \approx t\theta$; and
3) It remains to establish that $\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t' : \mathrm{T}$. Since t'_1 is ground and $t'_1 = t'$, it is implied by $\Gamma, x : \mathrm{T}_1; \mathcal{R} \sqcup \mathcal{R}'_1 \vdash_r t'_1 : \mathrm{T}$ relying on Lemmas 2 and 3.

It remains to deal with case c). According to Definition 5, we know that there exist b', m'_1, and m'_2 such that:

- $b' \approx b$, and $\Gamma'; \mathcal{R} \vdash_r b' : \mathrm{Bool}$; and
- $m_i \approx m'_i$, and $\Gamma'; \mathcal{R} \vdash_r m'_i : \mathrm{T}_i$ for $i = 1, 2$.

For $i = 1, 2$, we consider the substitution θ_i defined as follows:

$$\theta_i = \overline{\theta} \cup \{x \mapsto m'_i\}$$

where $\overline{\theta}$ is defined as in the previous case. In order to apply our induction hypothesis on each subtree with θ_1 (resp. θ_2), we have to verify that the hypotheses are satisfied. In particular, we have to establish that

$$\Gamma, x : \mathrm{T}_i; \mathcal{R} \vdash_r \theta_i.$$

First, we consider the case where $y \in \mathrm{dom}(\theta_i) \setminus \{x\}$, we have to show that $\Gamma, x : \mathrm{T}_i; \mathcal{R} \vdash_r \theta_i(y) : \Gamma(y)$, i.e. $\Gamma, x : \mathrm{T}_i; \mathcal{R} \vdash_r \theta(y) : \Gamma(y)$. This is proven using $\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \vdash_r \theta$, and $\theta(y)$ ground relying on Lemmas 2 and 3.

Now, we consider the case of the variable x. We have to show that $\Gamma, x : \mathrm{T}_i; \mathcal{R} \vdash_r m'_i : \mathrm{T}_i$. This is a consequence of $\Gamma'; \mathcal{R} \vdash_r m'_i : \mathrm{T}_i$ and Lemma 3.

Therefore, applying our induction hypothesis with θ_1 on the first subtree, and θ_2 on the second subtree, we deduce that there exist a ground term t'_i and $\mathcal{R}'_i$ (for $i = 1, 2$) such that:

1) $\mathrm{dom}(\mathcal{R}'_i) \subseteq R_i$;
2) $t\theta_i \approx t'_i$; and
3) $\Gamma, x : \mathrm{T}_i; \mathcal{R} \sqcup \mathcal{R}'_i \vdash_r t'_i : \mathrm{T}_i$.

Let $t' = \mathsf{if}\ b'\ \mathsf{then}\ t'_1\ \mathsf{else}\ t'_2$ and $\mathcal{R}' = \mathcal{R}'_1 \sqcup \mathcal{R}'_2$. We show that the requirements are satisfied:

1) $\mathrm{dom}(\mathcal{R}') = \mathrm{dom}(\mathcal{R}'_1) \sqcup \mathrm{dom}(\mathcal{R}'_2) \subseteq R_1 \sqcup R_2 = R$.
2) We have that

$$\begin{aligned} t' = \ & \mathsf{if}\ b'\ \mathsf{then}\ t'_1\ \mathsf{else}\ t'_2 \\ \approx \ & \mathsf{if}\ b'\ \mathsf{then}\ t\theta_1\ \mathsf{else}\ t\theta_2 \\ = \ & \mathsf{if}\ b'\ \mathsf{then}\ t\overline{\theta}[x \mapsto m'_1]\ \mathsf{else}\ t\overline{\theta}[x \mapsto m'_2] \\ \approx \ & t\overline{\theta}[x \mapsto \mathsf{if}\ b'\ \mathsf{then}\ m'_1\ \mathsf{else}\ m'_2] \\ \approx \ & t\overline{\theta}[x \mapsto \theta(x)] \\ \approx \ & t\theta \end{aligned}$$

3) It remains to establish that

$$\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \cup \mathcal{R}' \vdash_r t' : \mathrm{T}.$$

We have $\Gamma, x : \mathrm{T}_i; \mathcal{R} \sqcup \mathcal{R}'_i \vdash_r t'_i : \mathrm{T}_i$; and also $\Gamma'; \mathcal{R} \vdash_r b' : \mathrm{Bool}$. Relying on Lemmas 2 and 3, we deduce that $\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \cup \mathcal{R}' \vdash_r t'_i : \mathrm{T}_i$ and $\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2; \mathcal{R} \cup \mathcal{R}' \vdash_r b' : \mathrm{Bool}$. Then, we conclude using the rule IF.

Case of the rule SDEC:

$$\frac{\Gamma; R \vdash_b t : \mathrm{Msg} \quad \Gamma(k) = \mathrm{SK[T]}}{\Gamma; R \vdash_b \mathsf{sdec}(t, k) : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}}$$

By induction hypothesis, we know that there exist t'_0 and $\mathcal{R}'$ such that:

1) $\mathrm{dom}(\mathcal{R}') \subseteq R$;
2) $t\theta \approx t'_0$; and
3) $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t'_0 : \mathrm{Msg}$.

We note $\mathsf{senc}(p_i, k, r_i)$ the encryptions in t'_0 with k and random symbols from $\mathrm{dom}(\mathcal{R})$. Let

$$t' = \begin{cases} \mathsf{if}\ t'_0 = \mathsf{senc}(p_1, k, r_1)\ \mathsf{then}\ p_1\ \mathsf{else} \\ \quad \ldots \\ \quad \mathsf{if}\ t'_0 = \mathsf{senc}(p_n, k, r_n)\ \mathsf{then}\ p_n\ \mathsf{else}\ \mathsf{fail} \end{cases}$$

We have:

1) $\mathrm{dom}(\mathcal{R}') \subseteq R$;
2) By Lemma 16, we know that $\mathsf{sdec}(t'_0, k) \approx t'$, and thus:

$$t' \approx \mathsf{sdec}(t'_0, k) \approx \mathsf{sdec}(t\theta, k) \approx \mathsf{sdec}(t, k)\theta$$

3) It remains to establish that $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r t' : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}$. To do so, we first type the subterms of t', starting with the terms p_i. We know that $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r t'_0 : \mathrm{Msg}$, and each $\mathsf{senc}(p_i, k, r_i)$ is a subterm of t'_0. By Lemma 4, for each $i \in \{1, \ldots, n\}$, there exists some message type T'_i such that: $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r \mathsf{senc}(p_i, k, r_i) : \mathrm{T}'_i$.
There are three possible rules to derive a judgement of this form: FUN-LOW, FUN-MSG, and SENC. Since $r_i \in \mathrm{dom}(\mathcal{R})$, the rule SENC is actually the only one that can be used. It derives $\Gamma; \mathcal{R} \sqcup \mathcal{R}' \vdash_r \mathsf{senc}(p_i, k, r_i) : \mathrm{Low}$. Therefore, $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r p_i : \mathrm{T}$, and $\mathrm{T}'_i = \mathrm{Low}$ for each $i \in \{1, \ldots, n\}$.
Now, using the rule SUB-TYPING with $\mathrm{Low} \leq \mathrm{Msg}$, we can build the typing derivation for each test $t'_0 = \mathsf{senc}(p_i, k, r_i)$ occurring in t' using the rule EQ. For each $i \in \{1, \ldots, n\}$, we have that $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r p_i : \mathrm{T}$, and therefore $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r p_i : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}$. Lastly, we have also that $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r \mathsf{fail} : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}$, and we derive $\Gamma; \mathcal{R} \cup \mathcal{R}' \vdash_r t' : \mathrm{T} + \mathrm{Cst}^0_{\mathsf{fail}}$ using n times the IF.

This concludes the proof. □

APPENDIX E
TYPE SYSTEM FOR PROTOCOLS (META-LEVEL)

Let $\mathcal{P}$ be a protocol well-typed in the well-formed typing environment $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$, t be a meta-term built over $\mathcal{P}$ such that $\Gamma_{\mathcal{P}}; \emptyset \vdash t : \mathrm{T}$. The purpose of this section is to establish that for any trace model $\mathbb{T}$ of $\mathcal{P}$, we have that $\Gamma'; R' \vdash_b (t)^{\mathbb{T}}_{\mathcal{P}} : \mathrm{T}$ for some typing environment $(\Gamma'; R')$ that depends on the trace model $\mathbb{T}$ and that is formally defined below. This result is formally stated in Lemma 21 and is useful to establish our main Theorem 1 stated in Section IV, and proved at the end of this section.

For the remaining of this section, we assume given a protocol $\mathcal{P} = (\mathcal{A}\mathsf{ct}, \mathcal{U}_0, \prec)$ well-typed in $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$. We have therefore fixed the set $\mathcal{M}$ of macro symbols. Also, thanks to Definition 3, we know the existence of disjoint sets R_{A}, and $R^{\mathsf{s}}_{\mathsf{A}}$ for each $\mathsf{A} \in \mathcal{A}$, and $\mathsf{s} \in \mathcal{S}$. All these sets are subsets of $R_{\mathcal{P}}$.

We assume fixed the sets $\mathcal{D}_{\mathcal{I}}$, and $\mathcal{D}_{\mathcal{T}}$, as well as the ordering $<_{\mathcal{T}}$. We name *concrete index* an element of $\mathcal{D}_{\mathcal{I}}$, *concrete timestamp* an element of $\mathcal{D}_{\mathcal{T}}$, and *concrete macro* an object $\mathsf{m}[\vec{k}]@T$ with $\mathsf{m} \in \mathcal{M}$, $\vec{k} \in \mathcal{D}^*_{\mathcal{I}}$, and $T \in \mathcal{D}_{\mathcal{T}} \setminus \{\mathsf{undef}\}$. Therefore, we consider the instance of the base logic as defined in Section III-B and we denote $\mathcal{N}^{\mathcal{B}}$, $\mathcal{X}^{\mathcal{B}}$, $\mathscr{C}^{\mathcal{B}}$, $\mathcal{R}^{\mathcal{B}}$, $\mathcal{F}^{\mathcal{B}}$ the sets of names, variables, constants, randoms, and function symbols. We further assume $\mathscr{C}^{\mathcal{B}}$ is partitioned into $\mathscr{C}^{\mathcal{B},0} \cup \bigcup_{\mathsf{c} \in \mathcal{C}} \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$, where $\mathcal{C}$ is a finite set of identifiers. Moreover, for each concrete macro $\mathsf{m}[\vec{k}]@T$, we associate a variable in $\mathcal{X}^{\mathcal{B}}$ denoted $x_{\mathsf{m}[\vec{k}]@T}$. Note that, only state macros have a non-empty list of indices.

In addition to the total ordering $<_{\mathcal{T}}$, we fix a total ordering on state macros $\mathcal{S}$, and we extend it to the other macro symbols as follows. For any $\mathsf{s} \in \mathcal{S}$,

$$\mathsf{input} <_{\mathcal{M}} \mathsf{cond} <_{\mathcal{M}} \mathsf{exec} <_{\mathcal{M}} \mathsf{s} <_{\mathcal{M}} \mathsf{output} < \mathsf{frame}.$$

Given concrete macros $\mathsf{m}[\vec{k}]@T$, and $\mathsf{m}'[\vec{k'}]@T'$, we write $\mathsf{m}[\vec{k}]@T < \mathsf{m}'[\vec{k'}]@T'$ if

(i) $T \prec_{\mathcal{T}} T'$; or
(ii) $T = T'$ and $\mathsf{m} <_{\mathcal{M}} \mathsf{m}'$; or
(iii) $T = T'$ and $\mathsf{m} = \mathsf{m}'$ and $\vec{k} <_{\mathsf{lex}} \vec{k'}$.

Note that this last case occurs only when m and m' corresponds to the same state macro, and thus $\vec{k}$ and $\vec{k'}$ have the same length.

Given $(\Gamma; R)$ a well-formed typing environment built over the meta-logic under study, a concrete macro cm, and a trace model $\mathbb{T}_0 = (D_{\mathcal{I}}, D_{\mathcal{T}}, <_{\mathcal{T}}, \sigma^0_{\mathcal{I}}, \sigma^0_{\mathcal{T}})$ for some $\sigma^0_{\mathcal{I}}$ and $\sigma^0_{\mathcal{T}}$, we define the sets $\overline{\Gamma}^{\sim cm}$ with $\sim \in \{=, \leq, <\}$ as follows:

$$\begin{cases} \{(x_{\mathsf{m}@T}, \mathrm{Low}) \mid \mathsf{m} \in \{\mathsf{input}, \mathsf{frame}, \mathsf{output}\}, \mathsf{m}@T \sim cm\} \\ \cup \{(x_{\mathsf{m}@T}, \mathrm{Bool}) \mid \mathsf{m} \in \{\mathsf{cond}, \mathsf{exec}\}, \mathsf{m}@T \sim cm\} \\ \cup \{(x_{\mathsf{s}[\vec{k}]@T}, \mathrm{T}) \mid \mathsf{s} \in \mathcal{S}, \vec{k} \in \mathcal{D}^*_{\mathcal{I}}, \mathrm{T} = \Gamma(\mathsf{s}), \mathsf{s}[\vec{k}]@T \sim cm\} \\ \cup \{(\mathsf{n}_{\vec{j}}, \mathrm{T}) \mid \mathsf{n} \in \mathrm{dom}(\Gamma), \vec{j} \in D^*_{\mathcal{I}}, \mathrm{T} = \Gamma(\mathsf{n})\} \\ \cup \{(x, \mathrm{T}) \mid x \in \mathrm{dom}(\Gamma) \cap \mathcal{X}, \mathrm{T} = \Gamma(x)\} \end{cases}$$

Then, we define the set $\overline{R}^{=cm} \subseteq \mathcal{R}^{\mathcal{B}}$ which corresponds to the set of randoms introduced by the concrete macro $cm = \mathsf{m}[\vec{k}]@T_0$. This set is formally defined as follows[2]:

$$\begin{cases} \{\mathsf{r}_{\vec{\ell}} \mid \mathsf{r} \in R_{\mathsf{A}}\} \text{ when } \mathsf{m} = \mathsf{output},\ \vec{k} = \emptyset,\ T_0 = \mathsf{A}[\vec{\ell}] \\ \{\mathsf{r}_{\vec{\ell}} \mid \mathsf{r} \in R^{\mathsf{s}}_{\mathsf{A}}\} \text{ when } T_0 = \mathsf{A}[\vec{\ell}],\ \mathsf{m} = \mathsf{s}, \text{ and} \\ \quad \vec{k} = \sigma_{\mathcal{I}}(\vec{i}\,') \text{ with } u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]} = (\mathsf{if}\ \vec{j} =_{\iota} \vec{i}\,'\ \mathsf{then}\ \cdots) \\ \emptyset \qquad \text{otherwise} \end{cases}$$

Lastly, we define:

$$\overline{R_{\mathcal{P}}}^{\sim cm} = \bigcup_{cm' \sim cm} \overline{R_{\mathcal{P}}}^{=cm'} \text{ where } \sim \in \{\leq, <\}.$$

Note that, all these sets are defined w.r.t. a trace model $\mathbb{T}_0$ but only depend on $\mathcal{D}_{\mathcal{I}}$, $\mathcal{D}_{\mathcal{T}}$ and $<_{\mathcal{T}}$ and not on $\sigma^0_{\mathcal{I}}$ and $\sigma^0_{\mathcal{T}}$.

As the typing environment $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$ is well-formed, the typing environment $(\overline{\Gamma_{\mathcal{P}}}^{\sim cm}; \overline{R_{\mathcal{P}}}^{\sim cm})$ is well-formed too for any concrete macro cm, and $\sim \in \{=, <, \leq\}$. As in the restricted typing system, we may enrich the typing environment in a typing judgement $\vdash_b$. This is formally stated below, and can be proved as done in the restricted system.

Lemma 18. *Let t be a base-logic term and $(\Gamma; R)$ a well-formed typing environment such that $\Gamma; R \vdash_b t : \mathrm{T}$ for some type T. Consider $\Gamma' \supseteq \Gamma$ and $R' \supseteq R$ such that $(\Gamma'; R')$ is well-formed. We have that $\Gamma'; R' \vdash_b t : \mathrm{T}$.*

Lemma 19. *Let t be a base-logic term and $(\Gamma; R)$ a well-formed typing environment such that $\Gamma; R \vdash_b t : \mathrm{T}$ for some type T.*

- *For any variable x that does not appear in t, $\Gamma \setminus \{x\}; R \vdash_b t : \mathrm{T}$, where $\Gamma \setminus \{x\}$ is the environment obtained from Γ by removing the binding for x (if there was one).*
- *For any random symbol r that does not appear in t, $\Gamma; R \setminus \{r\} \vdash_b t : \mathrm{T}$.*

[2]Remember that by definition of a protocol, the update term $u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}$ is such that $\vec{i'}$ is a subset of $\vec{i}$.

Given a trace model $\mathbb{T} = (\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}, <_{\mathcal{T}}, \sigma_{\mathcal{I}}, \sigma_{\mathcal{T}})$ for some $\sigma_{\mathcal{I}}$ and $\sigma_{\mathcal{T}}$, and a meta-term t, we denote $\mu_{\mathbb{T}}(t)$ the maximal concrete macro cm w.r.t. $<$ such that x_{cm} occurs in $\overline{t}^{\mathbb{T}}$. If t does not contains any macro, we note $\mu_{\mathbb{T}}(t) = \bot$ with $\bot < cm$ for any cm, and we define $\overline{\Gamma_{\mathcal{P}}}^{\leq \bot} = \overline{\Gamma_{\mathcal{P}}}^{<\mathsf{input@init}}$, and $\overline{R_{\mathcal{P}}}^{\leq \bot} = \emptyset$.

Lemma 20. *Given a meta-term t built over $\mathcal{P}$, and $(\Gamma; R)$ be a well-formed typing environment such that $\Gamma; R \vdash t : \mathrm{T}$. Let $\mathbb{T} = (\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}, <_{\mathcal{T}}, \sigma_{\mathcal{I}}, \sigma_{\mathcal{T}})$ be a trace model of $\mathcal{P}$ for some $\sigma_{\mathcal{I}}$ and $\sigma_{\mathcal{T}}$. Let cm be such that $cm \geq \mu_{\mathbb{T}}(t)$. We have that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$ where:*

$$R' = \{\mathsf{r}_{\vec{k}} \mid \mathsf{r} \in R \text{ and } \mathsf{r}[\vec{i}] \in st(t) \text{ and } \sigma_{\mathcal{I}}(\vec{i}) = \vec{k}\}.$$

Proof. We establish this result by induction on the derivation witnessing $\Gamma; R \vdash t : \mathrm{T}$.

Case of the rule NAME*:* In such a case, we have that $t = \mathsf{n}[\vec{i}]$ for some n such that $(\mathsf{n} : \mathrm{T}) \in \Gamma$, and $\overline{t}^{\mathbb{T}} = \mathsf{n}_{\sigma_{\mathcal{I}}(\vec{i})}$. By definition of $\overline{\Gamma}^{\leq cm}$, we have that $\overline{\Gamma}^{\leq cm}(\mathsf{n}_{\sigma_{\mathcal{I}}(\vec{i})}) = \mathrm{T}$, and this allows us to conclude relying on the rule NAME.

Case of the rule VAR*:* In such a case, we have that $t = x$ for some x such that $\Gamma(x) = \mathrm{T}$, and $\overline{t}^{\mathbb{T}} = x$. By definition of $\overline{\Gamma}^{\leq cm}$, we have that $\overline{\Gamma}^{\leq cm}(x) = \mathrm{T}$, and this allows us to conclude relying on the rule VAR.

Case of the rules SUB-TYPING*,* FST*,* SND*, and* ZEROES*:* We conclude by relying on our induction hypothesis.

Case of the rule PAIR*:* In such a case, we have that $t = \langle t_1, t_2 \rangle$, $\mathrm{T} = \mathrm{T}_1 \times \mathrm{T}_2$, and $\overline{t}^{\mathbb{T}} = \langle \overline{t_1}^{\mathbb{T}}, \overline{t_2}^{\mathbb{T}} \rangle$. We have that $R = R_1 \sqcup R_2$, and $\Gamma; R_i \vdash t_i : \mathrm{T}_i$ for $i = 1, 2$. Relying on our induction hypothesis, as $cm \geq \mu_{\mathbb{T}}(t_i)$ for $i = 1, 2$, we know that $\overline{\Gamma}^{\leq cm}; R'_i \vdash_b \overline{t_i}^{\mathbb{T}} : \mathrm{T}_i$ for $i = 1, 2$. By definition of the set R'_1 and R'_2, and relying on the fact that R_1 and R_2 are disjoint, we have that R'_1 and R'_2 are disjoint too, and we have that $R' = R'_1 \sqcup R'_2$. This allows us to conclude that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$ by applying the rule PAIR.

The rules FUN-LOW, FUN-MSG, EQ, EQ-FALSE, EQ-TRUE-CST, EQ-FALSE-CST, and IF can be done in a rather similar way.

Case of the rule IF-TRUE*:* This case is rather similar to the previous one except that we have to enrich the set of randoms to obtain R'. Indeed, in such a case, we have that $t = \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$, and $\overline{t}^{\mathbb{T}} = \mathsf{if}\ \overline{t_0}^{\mathbb{T}}\ \mathsf{then}\ \overline{t_1}^{\mathbb{T}}\ \mathsf{else}\ \overline{t_2}^{\mathbb{T}}$. We have that $R = R_0 \sqcup R_1$, $\Gamma; R_0 \vdash t_0 : \mathrm{Cst}^0_{\mathsf{true}}$, and $\Gamma; R_1 \vdash t_1 : \mathrm{T}$. Relying on our induction hypothesis, we deduce that $\overline{\Gamma}^{\leq cm}; R'_0 \vdash_b \overline{t_0}^{\mathbb{T}} : \mathrm{Cst}^0_{\mathsf{true}}$, and $\overline{\Gamma}^{\leq cm}; R'_1 \vdash_b \overline{t_1}^{\mathbb{T}} : \mathrm{T}$. We also have that R'_0 and R'_1 are disjoint, and therefore we conclude that $\overline{\Gamma}^{\leq cm}; R'_0 \sqcup R'_1 \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$ by applying the rule IF-TRUE. We have that $R'_0 \sqcup R'_1 \subseteq R'$. The equality may not hold as some randoms may appear only in t_2. Therefore, we conclude relying on Lemma 18.

The case of the rule IF-FALSE is similar.

Case of the rule SENC*:* In such a case, $t = \mathsf{senc}(t_0, \mathsf{k}[\vec{j}], \mathsf{r}[\vec{i}])$, $\mathrm{T} = \mathrm{Low}$, $\Gamma(\mathsf{k}) = \mathrm{SK}[\mathrm{T}_0]$, and $R = R_0 \cup \{\mathsf{r}\}$. Moreover, we have that $\Gamma; R_0 \vdash t_0 : \mathrm{T}_0$, and $\overline{t} = \mathsf{senc}(\overline{t_0}, \mathsf{k}_{\sigma_{\mathcal{I}}(\vec{j})}, \mathsf{r}_{\sigma_{\mathcal{I}}(\vec{i})})$. The induction hypothesis give us: $\overline{\Gamma}^{\leq cm}; R'_0 \vdash_b \overline{t_0}^{\mathbb{T}} : \mathrm{T}_0$. By

definition of $\overline{\Gamma}^{\leq cm}$ and R', we know that $\overline{\Gamma}^{\leq cm}(\mathsf{k}_{\sigma_{\mathcal{I}}(\vec{i})}) = \mathrm{SK}[\mathrm{T}_0]$, and $\mathsf{r}_{\sigma_{\mathcal{I}}(\vec{i})} \in R'$. Applying the rule Senc, we deduce that

$$\overline{\Gamma}^{\leq cm}; R'_0 \cup \{\mathsf{r}_{\sigma_I(\vec{i})}\} \vdash_b \mathsf{senc}(\overline{t_0}, \mathsf{k}_{\sigma_I(\vec{j})}, \mathsf{r}_{\sigma_I(\vec{i})}) : \mathrm{Low}.$$

As $R' = R'_0 \cup \{\mathsf{r}_{\sigma_{\mathcal{I}}}(\vec{i})\}$, this allows us to conclude that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$.

The rule Sdec can be done in a similar way. It is even simpler as the Sdec does not involve any random.

Case of the rule Assign*:* We have that $t = t_0[x \mapsto t'_0]$, $R = R_1 \sqcup R_2$, $\Gamma, x : \mathrm{T}_0; R_1 \vdash t_0 : \mathrm{T}$ and $\Gamma; R_2 \vdash t'_0 : \mathrm{T}_0$. Applying our induction hypothesis, we deduce that:

- $\overline{\Gamma}^{\leq cm}, x : \mathrm{T}_0; R'_1 \vdash_b \overline{t_0}^{\mathbb{T}} : \mathrm{T}$, and
- $\overline{\Gamma}^{\leq cm}; R'_2 \vdash_b \overline{t'_0}^{\mathbb{T}} : \mathrm{T}_0$

Applying the rule Assign, this allows us to conclude that:

$$\overline{\Gamma}^{\leq cm}; R'_1 \sqcup R'_2 \vdash_b \overline{t_0}^{\mathbb{T}}[x \mapsto \overline{t'_0}^{\mathbb{T}}] : \mathrm{T}.$$

Actually, we have that:

$$R' = R'_1 \sqcup R'_2, \text{ and } \overline{t_0}^{\mathbb{T}}[x \mapsto \overline{t'_0}^{\mathbb{T}}] = \overline{t_0[x \mapsto t'_0]}^{\mathbb{T}}.$$

This allows us to conclude.

The rule Break-Sum can be done in a similar way.

Case of the rules Cst-0 *and* Cst-∞*:* In case of the rule Cst-0, we have that $t = \mathsf{c} \in \mathscr{C}^0$, $\mathrm{T} = \mathrm{Cst}^0_{\mathsf{c}}$, and $\overline{t}^{\mathbb{T}} = \mathsf{c} \in \mathscr{C}^0 \subseteq \mathscr{C}^{\mathcal{B}}$, thus the same rule applies and give us $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$. Now, in case of the rule Cst-∞, we have that $t = \mathsf{c} \in \mathscr{C}^{\infty}$. We have that $\overline{t}^{\mathbb{T}} = \mathsf{c}_{\sigma_{\mathcal{I}}(\vec{i})} \in \mathscr{C}^{\mathcal{B},\infty}_{\mathsf{c}}$, and applying the rule Cst-∞ of the base logic, we deduce that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \mathsf{c}_{\sigma_{\mathcal{I}}(\vec{i})} : \mathrm{Cst}^{\infty}_{\mathsf{c}}$. Hence, the result.

Case of the rules In*:* In such a case, we have that $\mathrm{T} = \mathrm{Low}$, $t = \mathsf{input}@T$, and thus $\overline{t}^{\mathbb{T}} = x_{\mathsf{input}@T'}$ with $T' = \overline{T}^{\mathbb{T}}$. By definition of $\overline{\Gamma}^{\leq cm}$, we have that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{Low}$ using the rule Var.

The rules Frame, Out, Cond, and Exec can be done in a similar way.

Case of the rule State*:* In such a case, we have that $\mathrm{T} = \Gamma(\mathsf{s})$, $t = \mathsf{s}[\vec{i}]@T$, and thus $\overline{t}^{\mathbb{T}} = x_{\mathsf{s}_{\sigma_{\mathcal{I}}(\vec{i})}@T'}$ with $T' = \overline{T}^{\mathbb{T}}$. By definition of $\overline{\Gamma}^{\leq cm}$, we have that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}$ using the rule Var.

Case of the rule Eq-Ind*:* In such a case, we have that $t = \vec{i} =_\iota \vec{j}$, $\overline{t}^{\mathbb{T}}$ is equal to true or false according to $\sigma_{\mathcal{I}}$, and $\mathrm{T} = \mathrm{Bool}$. We prove that $\overline{\Gamma}^{\leq cm}; R' \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{Bool}$ with the rule Cst-0, followed by Sub-Typing. This concludes the proof. □

Lemma 21. *Let $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$ be a well-formed typing environment, $\mathcal{P}$ be a protocol well-typed in $(\Gamma_{\mathcal{P}}; R_{\mathcal{P}})$, and t be a ground meta-term built over $\mathcal{P}$. Let $\mathbb{T} = (\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}, <_{\mathcal{T}}, \sigma_{\mathcal{I}}, \sigma_{\mathcal{T}})$ be a trace model of $\mathcal{P}$ for some σ_I and $\sigma_{\mathcal{T}}$ and such that $\overline{T}^{\mathbb{T}} \neq \mathsf{undef}$ for any $\mathsf{m}[\vec{i}]@T$ occurring in t. We have that:*

$$\Gamma_{\mathcal{P}}; \emptyset \vdash t : \mathrm{T} \text{ implies } \overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}}(t)}; \overline{R_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}}(t)} \vdash_b (t)^{\mathbb{T}}_{\mathcal{P}} : \mathrm{T}$$

Proof. We establish that for any trace model $\mathbb{T}'$ that coincides with $\mathbb{T}$ on $\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}$, and $<_{\mathcal{T}}$, for any ground meta-term t such that $\overline{T}^{\mathbb{T}'} \neq \mathsf{undef}$ for any $\mathsf{m}[\vec{i}]@T$ occurring in t, and any set R_0 such that $R_0 \cap \overline{R}^{\leq \mu_{\mathbb{T}'}(t)} = \emptyset$, we have that

$$\overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)}; R_0 \vdash_b \overline{t}^{\mathbb{T}'} : \mathrm{T} \text{ implies } \overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)}; R_0 \sqcup \overline{R_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)} \vdash_b (t)^{\mathbb{T}'}_{\mathcal{P}} : \mathrm{T}$$

by induction on $(\mu_{\mathbb{T}'}(t), <)$.

Note that, once this result is proved, then the result immediately follows relying on Lemma 20. Indeed, applying Lemma 20 on $\Gamma_{\mathcal{P}}; \emptyset \vdash t : \mathrm{T}$ with $\mathbb{T}$, and $\mu_{\mathbb{T}}(t)$, we deduce that

$$\overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}}(t)}; \emptyset \vdash_b \overline{t}^{\mathbb{T}} : \mathrm{T}.$$

Relying on the result stated above using $\mathbb{T}$, we then deduce that:

$$\overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}}(t)}; \overline{R_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}}(t)} \vdash_b (t)^{\mathbb{T}}_{\mathcal{P}} : \mathrm{T}.$$

Base case: The term $\overline{t}^{\mathbb{T}'}$ does not contain any variable representing a macro. Therefore, we have that $(t)^{\mathbb{T}'}_{\mathcal{P}} = \overline{t}^{\mathbb{T}'}$, and $\mu_{\mathbb{T}'}(t) = \bot$. By hypothesis, we have that $\overline{\Gamma_{\mathcal{P}}}^{\leq \bot}; R_0 \vdash_b \overline{t}^{\mathbb{T}'} : \mathrm{T}$, and thus we have that $\overline{\Gamma_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)}; R_0 \sqcup \overline{R_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)} \vdash_b (t)^{\mathbb{T}'}_{\mathcal{P}} : \mathrm{T}$.

Induction case: We perform a case analysis on the maximal (w.r.t. $<$ and considering the trace model $\mathbb{T}'$) macro occurring in t.

Case of an input *macro.* We denote $\mathsf{input}@T_0$ one of this occurrence of such a maximal macro, and $T = \overline{T_0}^{\mathbb{T}'} \in \mathcal{D}_{\mathcal{T}}$. Note that by hypothesis, we know that $T \neq \mathsf{undef}$. We consider the case where $T \neq \mathsf{init}$ but this case can be done in a rather similar way, since $\mathsf{input}@\mathsf{init}$ is defined as empty. We have that $\mu_{\mathbb{T}'}(t) = \mathsf{input}@T$. Let R_0 be such that $R_0 \cap \overline{R_{\mathcal{P}}}^{\leq \mu_{\mathbb{T}'}(t)} = \emptyset$. By hypothesis, we have that:

$$\overline{\Gamma_{\mathcal{P}}}^{\leq \mathsf{input}@T}; R_0 \vdash_b \overline{t}^{\mathbb{T}'} : \mathrm{T}.$$

We have also that:

$$\overline{\Gamma_{\mathcal{P}}}^{<\mathsf{input}@T}; \emptyset \vdash_b \mathsf{att}(x_{\mathsf{frame}@T'}) : \mathrm{Low}$$

where $T' = \mathsf{pred}_{\mathcal{T}}(T)$. This derivation is obtained using the rule Var (since $\overline{\Gamma_{\mathcal{P}}}^{<\mathsf{input}@T} = \overline{\Gamma_{\mathcal{P}}}^{\leq \mathsf{frame}@T'}$) and Fun-Low. Then, using the rule Assign, we obtain that:

$$\overline{\Gamma_{\mathcal{P}}}^{<\mathsf{input}@T}; R_0 \vdash_b \overline{t}^{\mathbb{T}'}[x_{\mathsf{input}@T} \mapsto \mathsf{att}(x_{\mathsf{frame}@T'})] : \mathrm{T}$$

Let t' be the meta-term obtained from t by replacing each occurrence of $\mathsf{input}@T''$ for some timestamps T'' such that $\overline{T''}^{\mathbb{T}'} = T$ by $\mathsf{att}(\mathsf{frame}@\mathsf{pred}(T_0))$. We have that:

$$\overline{t}^{\mathbb{T}'}[x_{\mathsf{input}@T} \mapsto \mathsf{att}(x_{\mathsf{frame}@T'})] = \overline{t'}^{\mathbb{T}'}$$

Since $T \neq \mathsf{undef}$ and $T \neq \mathsf{init}$, $\mathsf{frame}@T' < \mathsf{input}@T$, and thus $\overline{\Gamma_{\mathcal{P}}}^{\leq \mathsf{frame}@T'} = \overline{\Gamma_{\mathcal{P}}}^{<\mathsf{input}@T}$, we can apply our induction hypothesis to:

$$\overline{\Gamma_{\mathcal{P}}}^{\leq \mathsf{frame}@T'}; R_0 \vdash_b \overline{t'}^{\mathbb{T}'} : \mathrm{T}.$$

Note that $R_0 \cap \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t')} = \emptyset$ as $\overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t')} \subseteq \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)}$ and $R_0 \cap \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)} = \emptyset$ by hypothesis. Therefore, we deduce that

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t')}; R_0 \sqcup \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t')} \vdash_b (t')^{\mathbb{T}'}_\mathcal{P} : \mathrm{T}$$

Actually, we have that

$$\begin{aligned}(t')^{\mathbb{T}'}_\mathcal{P} &= \overline{t'}^{\mathbb{T}'}\theta \\ &= \overline{t}^{\mathbb{T}'}[x_{\mathsf{input@}T} \mapsto \mathsf{att}(x_{\mathsf{frame@}T'})]\theta \\ &= \overline{t}^{\mathbb{T}'}\theta \qquad \text{as } x_{\mathsf{input@}T}\theta = \mathsf{att}(x_{\mathsf{frame@}T'})\theta \\ &= (t)^{\mathbb{T}'}_\mathcal{P}\end{aligned}$$

Therefore, relying on Lemma 18, we conclude that

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)}; R_0 \sqcup \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)} \vdash_b (t)^{\mathbb{T}'}_\mathcal{P} : \mathrm{T}.$$

The case of a frame, cond or exec macro for which the unfolding does not introduce any random can be done in a similar way.

Case of an output *macro.* We denote $\mathsf{output@}T_0$ one of this occurrence of such a maximal macro, and $T = \overline{T_0}^{\mathbb{T}'} \in \mathcal{D}_\mathcal{T}$. Note that by hypothesis, we know that $T \neq \mathsf{undef}$. We consider the case where $T = \mathsf{A}[\vec{k}]$, and thus $T \neq \mathsf{init}$ but this case can be done in a rather similar way since $\mathsf{output@init}$ is defined as empty. We have that $\mu_{\mathbb{T}'}(t) = \mathsf{output@}T$. Let R_0 be such that $R_0 \cap \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)} = \emptyset$. By hypothesis, we have that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mathsf{output@}T}; R_0 \vdash_b \overline{t}^{\mathbb{T}'} : \mathrm{T}.$$

As we consider a well-typed protocol, we have that:

$$\Gamma_\mathcal{P}; R_\mathsf{A} \vdash \mathsf{if}\ \phi_{\mathsf{A}[\vec{i}]}\ \mathsf{then}\ o_{\mathsf{A}[\vec{i}]}\ \mathsf{else\ empty} : \mathrm{Low}$$

Let $t_{\mathsf{output}} = \mathsf{if}\ \phi_{\mathsf{A}[\vec{i}]}\ \mathsf{then}\ o_{\mathsf{A}[\vec{i}]}\ \mathsf{else\ empty}$. We consider w.l.o.g. that $\vec{i}$ does not appear inside t. We note $\mathbb{T}'' = \mathbb{T}'[\vec{i} \mapsto \vec{k}]$.

Applying Lemma 20 with $\mathbb{T}''$ and the element $\mu_{\mathbb{T}''}(t_{\mathsf{output}})$, we deduce that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}''}(t_{\mathsf{output}})}; R'_\mathsf{A} \vdash_b \overline{t_{\mathsf{output}}}^{\mathbb{T}''} : \mathrm{Low}$$

where $R'_\mathsf{A} \subseteq \{\mathsf{r}_{\vec{k}} \mid \mathsf{r} \in R_\mathsf{A}\}$. Then, using the rule Assign, and relying on Lemma 18, we obtain that:

$$\overline{\Gamma_\mathcal{P}}^{<\mathsf{output@}T}; R_0 \sqcup R'_\mathsf{A} \vdash_b \overline{t}^{\mathbb{T}'}[x_{\mathsf{output@}T} \mapsto \overline{t_{\mathsf{output}}}^{\mathbb{T}''}] : \mathrm{T}$$

Let t' be the meta-term obtained from t by replacing each occurrence of $\mathsf{output@}T''$ for some timestamps T'' such that $\overline{T''}^{\mathbb{T}'} = T$ by t_{output}. Since $\vec{i}$ does not appears in t, we have that $\overline{t}^{\mathbb{T}'} = \overline{t}^{\mathbb{T}''}$, and so $\overline{t}^{\mathbb{T}'}[x_{\mathsf{output@}T} \mapsto \overline{t_{\mathsf{output}}}^{\mathbb{T}''}] = \overline{t'}^{\mathbb{T}''}$. Using Lemma 19, we deduce that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}''}(t')}; R_0 \sqcup R'_\mathsf{A} \vdash_b \overline{t'}^{\mathbb{T}''} : \mathrm{T}$$

Applyin our induction hypothesis (with $\mathbb{T}''$), we deduce that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}''}(t')}; R_0 \sqcup R'_\mathsf{A} \sqcup \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}''}(t')} \vdash_b (t')^{\mathbb{T}''}_\mathcal{P} : \mathrm{T}.$$

Actually, we have that

$$\begin{aligned}(t')^{\mathbb{T}''}_\mathcal{P} &= \overline{t'}^{\mathbb{T}''}\theta \\ &= \overline{t}^{\mathbb{T}''}[x_{\mathsf{output@}T} \mapsto \overline{t_{\mathsf{output}}}^{\mathbb{T}''}]\theta \\ &= \overline{t}^{\mathbb{T}''}\theta \qquad \text{as } x_{\mathsf{output@}T}\theta = \overline{t_{\mathsf{output}}}^{\mathbb{T}''}\theta \\ &= (t)^{\mathbb{T}''}_\mathcal{P} \\ &= (t)^{\mathbb{T}'}_\mathcal{P}\end{aligned}$$

Therefore, relying on Lemma 18, we conclude that

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)}; R_0 \sqcup \overline{R_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(t)} \vdash_b (t)^{\mathbb{T}'}_\mathcal{P} : \mathrm{T}$$

This allows us to conclude.

Case of a state macro: $\mathsf{s}[\vec{\ell}]$. We denote $\mathsf{s}[\vec{j}]@T_0$ one of this occurrence of such a maximal macro, and $T = \overline{T_0}^{\mathbb{T}'} \in \mathcal{D}_\mathcal{T}$. Note that by hypothesis, we know that $T \neq \mathsf{undef}$. We have that $\mu_{\mathbb{T}'}(t) = \mathsf{s}[\vec{\ell}]@T$. Let R_0 be such that $R_0 \cap \overline{R_\mathcal{P}}^{\mu_{\mathbb{T}'}(t)} = \emptyset$. By hypothesis, we have that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mathsf{s}[\vec{\ell}]@T}; R_0 \vdash_b \overline{t}^{\mathbb{T}'} : \mathrm{T}.$$

We have three cases to consider. First, *(i)* the case $T = \mathsf{init}$. In other cases, $T = \mathsf{A}[\vec{k}]$, with:

$$u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]} = \mathsf{if}\ \vec{j} =_\iota \vec{i}'\ \mathsf{then}\ u\ \mathsf{else}\ \mathsf{s}[\vec{j}]@\mathsf{pred}(\mathsf{A}[\vec{i}])$$

In this case, we will consider w.l.o.g. that $\vec{i}$, $\vec{j}$ do not appear inside t, and note $\mathbb{T}'' = \mathbb{T}'[\vec{i} \mapsto \vec{k}, \vec{j} \mapsto \vec{\ell}]$. The two other cases are *(ii)* $\overline{\vec{j} =_\iota \vec{i}'}^{\mathbb{T}'} = \mathsf{false}$ and *(iii)* $\overline{\vec{j} =_\iota \vec{i}'}^{\mathbb{T}'} = \mathsf{true}$.

- Case *(i)*: By definition of a well-typed protocol, we have $R^\mathsf{s}_\mathsf{init} = \emptyset$, and thus $\Gamma_\mathcal{P}; \emptyset \vdash u_{\mathsf{init},\mathsf{s}[\vec{j}]} : \Gamma_\mathcal{P}(\mathsf{s})$. Applying Lemma 20 with $\mathbb{T}'$, we have that:

  $$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}'}(u_{\mathsf{init},\mathsf{s}[\vec{j}]})}; \emptyset \vdash_b \overline{u_{\mathsf{init},\mathsf{s}[\vec{j}]}}^{\mathbb{T}'} : \Gamma_\mathcal{P}(\mathsf{s})$$

  We can then conclude with the application of the rule Assign as in the case of an input macro.
- Case *(ii)*: In this case, we have that:

  $$\overline{u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}}^{\mathbb{T}''} = \mathsf{if\ false\ then}\ \overline{u}^{\mathbb{T}''}\ \mathsf{else}\ x_{\mathsf{s}[\vec{\ell}]@\mathsf{pred}(\mathsf{A}[\vec{k}])}.$$

  Notably with rule If-False, we get:

  $$\overline{\Gamma_\mathcal{P}}^{<\mathsf{s}[\vec{\ell}]@T}; \emptyset \vdash_b \overline{u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}}^{\mathbb{T}''} : \Gamma_\mathcal{P}(\mathsf{s})$$

  Then, we conclude with the application of the rule Assign as in the case of an input macro.
- Case *(iii)*: In this case, we have that:

  $$\overline{u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}}^{\mathbb{T}''} = \mathsf{if\ true\ then}\ \overline{u}^{\mathbb{T}''}\ \mathsf{else}\ x_{\mathsf{s}[\vec{\ell}]@\mathsf{pred}(\mathsf{A}[\vec{k}])}.$$

  By definition of a well-typed protocol, we have that

  $$\Gamma_\mathcal{P}; R^\mathsf{s}_\mathsf{A} \vdash u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]} : \Gamma_\mathcal{P}(\mathsf{s})$$

  Applying Lemma 20 with $\mathbb{T}''$, we have that:

  $$\overline{\Gamma_\mathcal{P}}^{\leq \mu_{\mathbb{T}''}(u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]})}; R' \vdash_b \overline{u_{\mathsf{A}[\vec{i}],\mathsf{s}[\vec{j}]}}^{\mathbb{T}''} : \Gamma_\mathcal{P}(\mathsf{s})$$

  with $R' \subseteq \{\mathsf{r}_{\vec{k}} \mid \mathsf{r} \in R^\mathsf{s}_\mathsf{A}\}$. Then, we conclude with the application of the rule Assign as in the case of an ouput macro.

This conclude the proof. □

We should be able now to prove our main result, *i.e.* Theorem 1, stated in Section IV. In fact, due to the presence

of states, the statement of the Theorem needs to be slightly modified, as follows.

Theorem 4. *Let $(\Gamma_\mathcal{P}; R_\mathcal{P})$ be a well-formed typing environment, $\mathcal{P}$ a protocol well-typed in $(\Gamma_\mathcal{P}; R_\mathcal{P})$, t_L and t_H be two meta-terms such that $\Gamma_\mathcal{P}; \emptyset \vdash t_L : \mathrm{Low}$ and $\Gamma_\mathcal{P}; \emptyset \vdash t_H : \mathrm{High}$. Let $\mathbb{T}$ be a trace model such that $\overline{T}^{\mathbb{T}} \neq \mathsf{undef}$ for any timestamp T occurring in t_L or t_H. Then $(t_L)^{\mathbb{T}}_\mathcal{P} \blacktriangleright (t_H)^{\mathbb{T}}_\mathcal{P}$.*

Compared to Theorem 1, the additional condition that all timestamps occurring in t_L, t_H must happen, *i.e.* not evaluate to undef, in the trace model $\mathbb{T}$. This condition is required for a technical reason: by definition, states at timestamp undef all evaluate to empty, which does not necessarily have the same type as that of the state. For that reason, we assume all timestamps in the terms happen, which is fine, since our definition of secrecy also requires it.

Proof. We have that $\Gamma_\mathcal{P}; \emptyset \vdash t_L : \mathrm{Low}$ and $\Gamma_\mathcal{P}; \emptyset \vdash t_H : \mathrm{High}$, and therefore we deduce that:

$$\Gamma_\mathcal{P}; \emptyset \vdash \langle t_L, t_H \rangle : \mathrm{Low} \times \mathrm{High}.$$

Let $\mathbb{T} = (\mathcal{D}_\mathcal{I}, \mathcal{D}_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$ be a trace model of $\mathcal{P}$ such that $\overline{T}^{\mathbb{T}} \neq \mathsf{undef}$ for any $\mathsf{m}[\vec{i}]@T$ occurring in t_L or t_H. By definition of being well-typed, we have that $\Gamma_\mathcal{P}$ does not contain variable, and therefore t_L and t_H are ground meta-terms. Applying Lemma 21, we deduce that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \overline{R_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)} \vdash_b (\langle t_L, t_H \rangle)^{\mathbb{T}}_\mathcal{P} : \mathrm{Low} \times \mathrm{High}$$

Applying Theorem 3, we deduce that:

$$\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models (\langle t_L, t_H \rangle)^{\mathbb{T}}_\mathcal{P} : \mathrm{Low} \times \mathrm{High}$$

for some $\mathcal{R}$.

Applying Definition 5, we know that:

- $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models \mathsf{fst}((\langle t_L, t_H \rangle)^{\mathbb{T}}_\mathcal{P}) : \mathrm{Low}$, and
- $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models \mathsf{snd}((\langle t_L, t_H \rangle)^{\mathbb{T}}_\mathcal{P}) : \mathrm{High}$,

and thus, relying on Lemma 9, we have that:

- $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models (t_L)^{\mathbb{T}}_\mathcal{P} : \mathrm{Low}$, and
- $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models (t_H)^{\mathbb{T}}_\mathcal{P} : \mathrm{High}$.

Relying again on Definition 5, we know that for all t'_L such that $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models t'_L : \mathrm{Low}$, we have that $t'_L \blacktriangleright (t_H)^{\mathbb{T}}_\mathcal{P}$. We have that $\overline{\Gamma_\mathcal{P}}^{\leq \mu_\mathbb{T}(\langle t_L, t_H \rangle)}; \mathcal{R} \models (t_L)^{\mathbb{T}}_\mathcal{P} : \mathrm{Low}$, and thus we deduce that $(t_L)^{\mathbb{T}}_\mathcal{P} \blacktriangleright (t_H)^{\mathbb{T}}_\mathcal{P}$, *i.e.* for any computational model $\mathbb{M}$, and any PPTM $\mathcal{A}$, we have that:

$$\mathbb{P}_\rho \left[\mathcal{A}(1^\eta, \rho_a, [\![(t_L)^{\mathbb{T}}_\mathcal{P}]\!]^{\mathbb{M}}(1^\eta, \rho)) = [\![(t_H)^{\mathbb{T}}_\mathcal{P}]\!]^{\mathbb{M}}(1^\eta, \rho) \right] \in \mathsf{negl}(\eta).$$

This concludes the proof. □

Appendix F
Review of our case studies (symmetric protocols)

We provide below a short description of each protocol that we have analysed, as well as a partial description of the well-formed typing environment under which we succeed to perform the typing. Below, we do not make explicit the way we add parentheses around the $+$ operator as it does not really matter. Regarding parentheses around the $\times$ operator, we implicitly assume that $\mathrm{T}_1 \times \mathrm{T}_2 \times \mathrm{T}_3$ means $\mathrm{T}_1 \times (\mathrm{T}_2 \times \mathrm{T}_3)$ similarly to our pairing operator.

Denning-Sacco: The tagged version of the protocol we consider is as follows:

$$\begin{array}{ll} A \to S: & A, B \\ S \to A: & \{B, K_{ab}, \{\mathsf{resp}, K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\ A \to B: & \{\mathsf{resp}, K_{ab}, A\}_{K_{bs}} \end{array}$$

With the explicit tag resp, this protocol can be typed when considering an environment such that:

$$\begin{aligned} \mathsf{k_s} : \mathrm{SK}[&\mathrm{Cst}^\infty_\mathsf{a} \times \mathrm{High} \times \mathrm{Low} + \mathrm{Cst}^0_\mathsf{resp} \times \mathrm{High} \times \mathrm{Cst}^\infty_\mathsf{a} + \\ &\mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}} \times \mathrm{Low} \times \mathrm{Low} + \mathrm{Cst}^0_\mathsf{resp} \times \mathrm{Low} \times \mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}}] \end{aligned}$$

This allows us to give a precise type to each plaintext contained in an encryption with a key $\mathsf{k_s}[i]$, and therefore to infer that the established key has type High when needed.

Needham-Schroeder*: As we do not model the challenge-response mechanism, the Needham-Schroeder protocol is actually rather similar to the Denning-Sacco protocol, and we have to consider its tagged version to be able to type it and establish secrecy of the session key.

$$\begin{array}{ll} A \to S: & A, B, N_a \\ S \to A: & \{B, N_a, K_{ab}, \{\mathsf{resp}, K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\ A \to B: & \{\mathsf{resp}, K_{ab}, A\}_{K_{bs}} \end{array}$$

This protocol can be typed when considering an environment such that $\mathsf{k_s}$ is typed as follows:

$$\begin{aligned} \mathrm{SK}[&\mathrm{Cst}^\infty_\mathsf{a} \times \mathrm{Low} \times \mathrm{High} \times \mathrm{Low} + \mathrm{Cst}^0_\mathsf{resp} \times \mathrm{High} \times \mathrm{Cst}^\infty_\mathsf{a} \\ &+ \mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}} \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Low} + \mathrm{Cst}^0_\mathsf{resp} \times \mathrm{Low} \times \mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}}] \end{aligned}$$

Otways-Rees: The tagged version of the Otway-Rees protocol can be informally described as follows.

$$\begin{array}{l} A \to B : M, A, B, \{\mathsf{req}, N_a, M, A, B\}_{K_{as}} \\ B \to S : A, B, \{\mathsf{req}, N_a, M, A, B\}_{K_{as}}, \{\mathsf{req}, N_b, M, A, B\}_{K_{bs}} \\ S \to B : \{\mathsf{rep}, N_a, K_{ab}\}_{K_{as}}, \{\mathsf{rep}, N_b, K_{ab}\}_{K_{bs}} \\ B \to A : \{\mathsf{rep}, N_a, K_{ab}\}_{K_{as}} \end{array}$$

Note that, it is actually *not* possible to provide a typing environment allowing us to type its untagged version, *i.e.* without req and rep (together with terms t^X_H). The absence of this typing environment actually corresponds to the existence of an attack due to the possible confusion between ciphertexts that allows the intiator (and the responder) to accept $\langle M, A, B \rangle$ as a key. Establishing the secrecy of the key as seen by S is actually not an issue, even when considering the untagged version of the protocol, and can be done in a typing system where: $\mathsf{k_s} : \mathrm{SK}[\mathrm{Msg}]$.

Otherwise, to be able to type both the protocol, and the terms t^X_H expressing the secrecy properties, the following type is required for $\mathsf{k_s}$:

$$\begin{aligned} \mathsf{k_s} : \mathrm{SK}[&\mathrm{Cst}^0_\mathsf{req} \times \mathrm{High} \times \mathrm{Low} \times \mathrm{Cst}^\infty_\mathsf{a} \times \mathrm{Cst}^\infty_\mathsf{a} + \\ &\mathrm{Cst}^0_\mathsf{req} \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}^\infty_\mathsf{a} \times \mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}} + \\ &\mathrm{Cst}^0_\mathsf{req} \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}^\infty_{\mathsf{a}^\mathsf{d}} \times \mathrm{Cst}^\infty_\mathsf{a} + \\ &\mathrm{Cst}^0_\mathsf{rep} \times \mathrm{High} \times \mathrm{High} + \mathrm{Cst}^0_\mathsf{rep} \times \mathrm{Low} \times \mathrm{Low}] \end{aligned}$$

Yahalom*: Consider now the original Yahalom protocol. Its tagged version (without the last ciphertext) can be informally described as follows.

$$\begin{array}{l} A \to B : A, N_a \\ B \to S : B, \{1, A, N_a, N_b\}_{K_{bs}} \\ S \to A : \{2, B, K_{ab}, N_a, N_b\}_{K_{as}}, \{3, A, K_{ab}\}_{K_{bs}} \\ A \to B : \{3, A, K_{ab}\}_{K_{bs}} \end{array}$$

Again, tags are needed to avoid confusion. Moreover, as the nonce N_a is sent in clear in the first message, it is important that na (and also na^d) is typed Low, and typing similarly nb and nb^d will allow us to type the protocol, and the terms t^{x}_H expressing secrecy using the following type for $\mathsf{k_s}$:

$$\begin{array}{l} \mathrm{SK}[\mathrm{Cst}_1^0 \times \mathrm{Low} \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{High} \times \mathrm{Low} \ + \ \mathrm{Cst}_3^\infty \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{High} \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \times \mathrm{Low} \times \mathrm{Low} \ + \ \mathrm{Cst}_3^\infty \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \times \mathrm{Low} \ ] \end{array}$$

The type of the key $\mathsf{k_s}$ reflects the fact that ciphertexts tagged with 1 do not contain any secret information, and therefore the resulting plaintexts can be leaked to the attacker.

Yahalom-Paulson*: Similarly, the Yahalon-Paulson protocol need to be explicitly tagged in order to be shown to be well-typed. The last ciphertext, a key confirmation step, has been removed to perform the analysis.

$$\begin{array}{l} A \to B : A, N_a \\ B \to S : B, N_b, \{1, A, N_a\}_{K_{bs}} \\ S \to A : N_b, \{2, B, K_{ab}, N_a\}_{K_{as}}, \{3, A, B, K_{ab}, N_b\}_{K_{bs}} \\ A \to B : \{3, A, B, K_{ab}, N_b\}_{K_{bs}} \end{array}$$

As the previous protocol, this protocol can be typed using Low for typing nonces, and the following type for $\mathsf{k_s}$.

$$\begin{array}{l} \mathrm{SK}[\mathrm{Cst}_1^0 \times \mathrm{Low} \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{High} \times \mathrm{Low} \ + \ \mathrm{Cst}_2^0 \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \times \mathrm{Low} \\ \quad + \mathrm{Cst}_3^0 \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{Low} \times \mathrm{High} \times \mathrm{Low} \\ \quad + \mathrm{Cst}_3^0 \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \times \mathrm{Low} \ ] \end{array}$$

Mechanism 6: This protocol does not rely on a server to establish the session key, but assume the existence of a symmetric key between the two participants A and B.

$$\begin{array}{l} B \to A : R_b \\ A \to B : \{1, R_a, R_b, B, K_a\}_{K_{ab}} \\ B \to A : \{2, R_b, R_a, K_b\}_{K_{ab}} \end{array}$$

Then, the fresh symmetric key is derived using a key derivation function taking as input K_a and K_b, and at least the random numbers R_a and R_b. Here, we consider weak secrecy of K_a (resp. K_b) from the point of view of both A and B. This can be established considering the following type for the long-term key k shared between honest agents:

$$\begin{array}{ll} \mathsf{k}: & \mathrm{SK}[\mathrm{Cst}_1^0 \times \mathrm{High} \times \mathrm{Low} \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{High} \\ & \quad + \mathrm{Cst}_2^0 \times \mathrm{Low} \times \mathrm{High} \times \mathrm{High} \ ] \end{array}$$

Mechanism 9: This protocol does rely on a server that is responsible of the generation of the session key K.

$$\begin{array}{l} B \to A : R_b \\ A \to S : R_a, R_b, B \\ S \to A : \{1, R_a, K, B\}_{K_{as}}, \{2, R_b, K, A\}_{K_{bs}} \\ A \to B : \{2, R_b, K, A\}_{K_{bs}} \end{array}$$

This protocol can be typed using Low for nonces, and the following type for $\mathsf{k_s}$:

$$\begin{array}{l} \mathrm{SK}[\mathrm{Cst}_1^0 \times \mathrm{Low} \times \mathrm{High} \times \mathrm{Cst}_\mathsf{a}^\infty \\ \quad + \mathrm{Cst}_1^0 \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Low} \times \mathrm{High} \times \mathrm{Cst}_\mathsf{a}^\infty \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty] \end{array}$$

Mechanism 13: This protocol does rely on a server but the session key K is generated by one of the participant of the protocol (here A).

$$\begin{array}{l} B \to A : R_b \\ A \to S : \{1, R_a, R_b, B, K\}_{K_{as}} \\ S \to A : \{2, R_a, R_b, B\}_{K_{as}}, \{3, R_b, K, A\}_{K_{bs}} \\ A \to B : \{3, R_b, K, A\}_{K_{bs}} \end{array}$$

Again, this protocol can be typed using Low for nonces, and the following type for $\mathsf{k_s}$:

$$\begin{array}{l} \mathrm{SK}[\mathrm{Cst}_1^0 \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}_\mathsf{a}^\infty \times \mathrm{High} \\ \quad + \mathrm{Cst}_1^0 \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty \times \mathrm{Low} \\ \quad + \mathrm{Cst}_2^0 \times \mathrm{Low} \\ \quad + \mathrm{Cst}_3^0 \times \mathrm{Low} \times \mathrm{High} \times \mathrm{Cst}_\mathsf{a}^\infty \\ \quad + \mathrm{Cst}_3^0 \times \mathrm{Low} \times \mathrm{Low} \times \mathrm{Cst}_{\mathsf{a}^\mathsf{d}}^\infty] \end{array}$$

Regarding mechanisms 9 and 13, an optional key confirmation exchange has not been modelled. We may also note that the tags used in the norm are distinct constants per protocol meaning that a similar analysis (with similar conclusions) would have been possible considering all the mechanisms together.

APPENDIX G
DEALING WITH ASYMMETRIC ENCRYPTION

In this section, we showcase the extensibility of our type system by adding support for asymmetric encryption. We assume three new function symbols: aenc and adec, interpreted respectively as the encryption and decryption algorithms of an asymmetric scheme, and pk that produces the public encryption key associated to a private key. The encryption scheme is assumed to satisfy the IND-CCA-2 property (see Appendix A-A).

A. Some extra rules for the type system

We add a new type for asymmetric private keys: AK[T]. Similarly to the symmetric case, the type of a private key indicates the type of plaintexts intended to be encrypted with the associated public key, which provides information when decrypting. Public encryption keys, on the other hand, are typed Low. We consider three extra rules:

$$\frac{\Gamma(\mathsf{k}) = \mathrm{AK}[\mathrm{T}]}{\Gamma; R \vdash \mathsf{pk}(\mathsf{k}[\vec{j}]) : \mathrm{Low}}\ \textsc{Pk}$$

$$\frac{\Gamma; R \vdash t : \mathrm{T} \qquad \Gamma(\mathsf{k}) = \mathrm{AK}[\mathrm{T}]}{\Gamma; R \sqcup \{\mathsf{r}\} \vdash \mathsf{aenc}(t, \mathsf{pk}(\mathsf{k}[\vec{j}]), \mathsf{r}[\vec{i}]) : \mathrm{Low}}\ \textsc{Aenc}$$

$$\frac{\Gamma; R \vdash t : \text{Msg} \qquad \Gamma(\mathsf{k}) = \text{AK[T]}}{\Gamma; R \vdash \mathsf{adec}(t, \mathsf{k}[\vec{j}]) : \text{T} + \text{Low}}\ \textsc{Adec}$$

Except for the possibility to publish the encryption key, the main difference with the symmetric case is the decryption rule. As the encryption key is public, the attacker can use it to encrypt messages of his own. Decryption hence returns either an honest message (type T) or a public one, *i.e.* of type Low. This creates an additional difficulty compared to symmetric case: when decrypting a ciphertext, if it was constructed by the attacker, there may not always exist a term that represents the decrypted message. That requires some work, since the interpretation of $\text{T} + \text{Low}$ mandates the existence of an equivalent term typing Low in the restricted system.

To find a term for public results of decryptions, we add in the base logic a new ternary function symbol $\mathsf{adec}^\star$ with a fixed semantics: $\mathsf{adec}^\star$ takes a ciphertext t, a private key k and a list of ciphertexts. Lists are represented by nested pairs $\langle t_1, \langle t_2, \ldots, \langle t_{n-1}, t_n \rangle \ldots \rangle \rangle$, and are noted $\langle t_i \rangle_{1 \leq i \leq n}$. We define $[\![\mathsf{adec}^\star(t, k, \langle t_i \rangle_{1 \leq i \leq n})]\!]^{\mathbb{M}}(1^\eta, \rho)$ as:

- $[\![\mathsf{fail}]\!]^{\mathbb{M}}(1^\eta, \rho)$ if $[\![t]\!]^{\mathbb{M}}(1^\eta, \rho) = [\![t_i]\!]^{\mathbb{M}}(1^\eta, \rho)$ for some $i \in \{1, \ldots, n\}$
- $[\![\mathsf{adec}(t, k)]\!]^{\mathbb{M}}(1^\eta, \rho)$ otherwise.

As for symmetric encryption, we conduct the proof in two steps. Symbols adec are removed and replaced with $\mathsf{adec}^\star$, similarly to how we removed sdec using the IND-CTXT assumption. Then, in the restricted system, we only need to handle $\mathsf{adec}^\star$ instead of adec.

B. Restricted type system

Due to the peculiar semantics of $\mathsf{adec}^\star$, the associated typing rule is rather complex. Indeed, we must check that the list of forbidden ciphertexts contains all possible protocol-generated ciphertexts in the term. In that case, since $\mathsf{adec}^\star$ prevents the decryption of ciphertexts in its argument, it only decrypts adversarially generated ciphertexts, and its result is thus of type Low.

$$\frac{\Gamma(k) = \text{AK[T]}}{\Gamma; \mathcal{R} \vdash_r \mathsf{pk}(k) : \text{Low}}\ \textsc{Pk}$$

$$\frac{\Gamma; \mathcal{R} \vdash_r t : \text{T} \qquad \Gamma(k) = \text{AK[T]} \qquad \mathcal{R}(r) = (t, \mathsf{pk}(k))}{\Gamma; \mathcal{R} \vdash_r \mathsf{aenc}(t, \mathsf{pk}(k), r) : \text{Low}}\ \textsc{Aenc}$$

$$\frac{\Gamma; R \vdash_r t : \text{Msg} \qquad \Gamma(k) = \text{AK[T]}}{\Gamma; R \vdash_r \mathsf{adec}^\star(t, k, \langle t_i \rangle_{1 \leq i \leq n}) : \text{Low}}\ \textsc{Adec}^\star$$

where $\langle t_i \rangle_{1 \leq i \leq n}$ is the sequence of all the ciphertexts (terms headed with symbol aenc) using $\mathsf{pk}(k)$ as a key, using an element of $\mathcal{R}^{\mathcal{B}}$ as random, and occurring as a subterm in t.

The soundness proofs for the restricted type system must be adapted to handle these additional rules. In most cases, this is simply a matter of adding another case to the inductions, which is proved similarly to the symmetric case – we will not detail these adjustments. Some parts, however, are modified more significantly. We identify three important changes. The definition of the evaluator for terms, and the associated Lemma 8, needs to be adapted to evaluate asymmetric operations. The sub-section dealing with the cryptographic arguments must of course be extended. Lastly, Lemma 6 also needs adjustments: a subterm typing High can now also be contained under an asymmetric encryption. The proof and the use of this modified lemma in the main theorem of the section are similar to the symmetric case, so we do not detail them.

Evaluator. We have to modify the definition of the evaluator and its oracle to handle the function symbols aenc and $\mathsf{adec}^\star$.

Definition 18 (Modification of Definition 9)**.** *We give the evaluator access to three new oracles, in addition to the one for names and the one for symmetric encryption, to handle asymmetric encryption and decryption, and public keys. For simplicity, we call $\mathcal{O} = \{\mathcal{O}_{\mathsf{name}}, \mathcal{O}_{\mathsf{enc}}, \mathcal{O}_{\mathsf{aenc}}, \mathcal{O}_{\mathsf{adec}}, \mathcal{O}_{\mathsf{pk}}\}$ the set of five total oracles, and see it as a single oracle when there is no ambiguity as to which oracle is called.*

- $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{pk}(k))(1^\eta, \rho_{\mathsf{a}})$ *calls* $\mathcal{O}_{\mathsf{pk}}(\text{“}k\text{”})$ *and returns its answer.*
- $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{aenc}(m, \mathsf{pk}(k), r))(1^\eta, \rho_{\mathsf{a}})$ *first checks that $r \in \mathrm{dom}(\mathcal{R})$, and if so computes the plaintext $p = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(m)(1^\eta, \rho_{\mathsf{a}})$, and then calls $\mathcal{O}_{\mathsf{aenc}}(p, \text{“}k\text{”}, \text{“}r\text{”})$ and returns its answer.*
- *if $\Gamma(k) = \text{AK[T]}$, $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{adec}^\star(t, k, \langle t_i \rangle_{1 \leq i \leq \ell})(1^\eta, \rho_{\mathsf{a}})$ computes $c = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}})$ and $\langle c_i \rangle_{1 \leq i \leq \ell} = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\langle t_i \rangle_{1 \leq i \leq \ell})(1^\eta, \rho_{\mathsf{a}})$. If the bitstring c is equal to one of the c_i, then the evaluator returns $[\![\mathsf{fail}]\!]^{\mathbb{M}}(1^\eta)$. Otherwise, it calls $\mathcal{O}_{\mathsf{adec}}(c, \text{“}k\text{”})$ and returns its answer.*

The three new oracles behave as follows.

- *for any k typed AK[T] in Γ, $\mathcal{O}_{\mathsf{pk}}(\text{“}k\text{”})$ returns $[\![\mathsf{pk}(k)]\!]^{\mathbb{M}}(1^\eta, \rho)$ and fails otherwise;*
- *for any random r, asymmetric key k, and bitstring m, the oracle $\mathcal{O}_{\mathsf{aenc}}(m, \text{“}k\text{”}, \text{“}r\text{”})$ checks whether $\mathcal{R}(r) = (m', \mathsf{pk}(k))$ for some m' such that $m = [\![m']\!]^{\mathbb{M}}(1^\eta, \rho)$. If so, it computes $c = [\![\mathsf{aenc}]\!]^{\mathbb{M}}(m, [\![\mathsf{pk}(k)]\!]^{\mathbb{M}}(1^\eta, \rho), [\![r]\!]^{\mathbb{M}}(1^\eta, \rho))$ and returns it. Otherwise, it fails.*
- *for any asymmetric key k, and bitstring t, the oracle $\mathcal{O}_{\mathsf{adec}}(c, \text{“}k\text{”})$ returns $[\![\mathsf{adec}]\!]^{\mathbb{M}}(c, [\![k]\!]^{\mathbb{M}}(1^\eta, \rho))$.*

It might seem surprising that we include an oracle for asymmetric encryption, given that the evaluator already has access to all public keys through $\mathcal{O}_{\mathsf{pk}}$. Oracle $\mathcal{O}_{\mathsf{aenc}}$ is still needed, as we do not wish to give the evaluator direct access to the encryption randomness. Contrary to the symmetric case, we need to include the decryption oracle $\mathcal{O}_{\mathsf{adec}}$, in order to evaluate terms that use $\mathsf{adec}^\star$. The evaluator however only calls that oracle when the ciphertext is “authorised” by $\mathsf{adec}^\star$.

Now, we must adjust the proof of Lemma 8, showing that well-typed terms are evaluable. There are some subtle changes, and thus we present it in detail.

Lemma 8. *Let $(\Gamma; \mathcal{R})$ be a well-formed mapping environment, t a ground base-logic term such that $\Gamma; \mathcal{R} \vdash_r t : \text{T}$ for some message type T. We have that t is evaluable in $(\Gamma; \mathcal{R})$.*

Proof. This proof is done by induction on the typing tree. We detail here the cases for the additional rules used for asymmetric encryption.

In the case of rule PK, k has type $\mathsf{AK}[\mathrm{T}']$, so the oracle does not fail and sends the correct value.

The case of rule AENC is similar to SENC.

Finally, the case of rule ADEC* is slightly more involved. In that case, $\mathrm{T} = \mathsf{Msg}$, $\Gamma(k) = \mathsf{AK}[\mathrm{T}']$ for some T', and $t = \mathsf{adec}^\star(t', k, \langle t_i \rangle_{1 \le i \le \ell})$, where t_i are all ciphertexts (terms headed with symbol aenc) using $\mathsf{pk}(k)$ as a key, using an element of $\mathcal{R}^{\mathcal{B}}$ as random, and occurring as a subterm of t'.

First, the evaluator computes $c = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}})$ and $c_i = \mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(t_i)(1^\eta, \rho_{\mathsf{a}})$ for $1 \le i \le \ell$. Since, by the premise of the rule, $\Gamma; \mathcal{R} \vdash_r t' : \mathsf{Msg}$, we have by induction hypothesis $\mathbb{P}_\rho \left[c = [\![t']\!]^{\mathbb{M}}(1^\eta, \rho) \right] \in \mathsf{ow}(\eta)$. We deal with the t_i similarly. Since each t_i is a subterm of t (but neither a name nor a random), by Lemma 4, we get $\Gamma; \mathcal{R} \vdash_r t_i : \mathrm{T}''$ for some T'' (as a subtree of the derivation witnessing $\Gamma; \mathcal{R} \vdash_r t : \mathrm{T}$). Thus, we can apply the induction hypothesis to it, and we have $\mathbb{P}_\rho \left[c_i = [\![t_i]\!]^{\mathbb{M}}(1^\eta, \rho) \right] \in \mathsf{ow}(\eta)$ for each i.

Consider the probability of c and each c_i all being correctly evaluated. Since ℓ is fixed, independent of ρ, η (it only depends on the structure of term t), that probability is overwhelming. From now on consider only the ρ for which that event occurs.

The evaluator then checks if $c = c_i$ for any i.

- If so, it returns $[\![\mathsf{fail}]\!]^{\mathbb{M}}(1^\eta)$. In that case, we also have $[\![t']\!]^{\mathbb{M}}(1^\eta, \rho) = [\![t_i]\!]^{\mathbb{M}}(1^\eta, \rho)$, and $\mathsf{adec}^\star$ also fails: $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}(\mathsf{adec}^\star(t', k, \langle t_i \rangle_{1 \le i \le \ell}))(1^\eta, \rho_{\mathsf{a}}) = [\![\mathsf{adec}^\star(t', k, \langle t_i \rangle_{1 \le i \le \ell})]\!]^{\mathbb{M}}(1^\eta)$.
- Otherwise, $\mathsf{adec}^\star$ successfully decrypts t', and $[\![\mathsf{adec}^\star(t', k, \langle t_i \rangle_{1 \le i \le \ell})]\!]^{\mathbb{M}}(1^\eta) = [\![\mathsf{adec}(t', k)]\!]^{\mathbb{M}}(1^\eta, \rho)$. The evaluator on the other hand calls the oracle $\mathcal{O}_{\mathsf{adec}}$, which computes the same value.

Thus, with overwhelming probability, the evaluator successfully computes $[\![t]\!]^{\mathbb{M}}(1^\eta, \rho)$, which concludes the proof. □

Cryptography. The previous proofs, in the symmetric game reductions, involve replicating the behaviour of $\mathcal{E}^{\mathcal{O}}_{\mathbb{M},\Gamma,\mathcal{R}}$ using the oracles given by the games IND-CPA or INT-CTXT. We must extend these proofs to simulate the additional oracles. This is straightforward, since the attacker in these games has access to any name except the symmetric key used for the challenge, and any nonce used as encryption randomness with that key. Therefore, the attacker has access to the private asymmetric key, and can easily use it to simulate $\mathcal{O}_{\mathsf{aenc}}, \mathcal{O}_{\mathsf{adec}}, \mathcal{O}_{\mathsf{pk}}$.

Now, more importantly, we have also to deal with asymmetric encryptions. For that, similarly to δ_c in the symmetric case, we define δ^a_c for an asymmetric ciphertext c, such that, in particular, $\delta^a_{\mathsf{aenc}(m,\mathsf{pk}(k),r)}(\mathsf{aenc}(m, \mathsf{pk}(k), r)) = \mathsf{aenc}(\mathsf{zeros}(m), \mathsf{pk}(k), r)$. The statement and the proof of Lemma 12 remain unchanged. We then prove the following lemma, analogous to Lemma 13.

Lemma 22. *Consider two ground base logic terms t, t', a well-typed mapping environment $(\Gamma; \mathcal{R})$ such that $\Gamma; \mathcal{R} \vdash_r t : \mathsf{Msg}$ and $\Gamma; \mathcal{R} \vdash_r t' : \mathsf{Msg}$. Consider a term $c = \mathsf{aenc}(m, \mathsf{pk}(k), r)$, such that $\mathcal{R}(r) = (m, k)$. We have that:*

$$\delta^a_c(t) \blacktriangleright \delta^a_c(t') \text{ implies } t \blacktriangleright t'.$$

Proof. The proof is similar to the one for Lemma 13. It consists in reducing the IND-CCA-2 assumption on the encryption primitive.

Assume $\delta^a_c(t) \blacktriangleright \delta^a_c(t')$. Fix a computational model $\mathbb{M}$ and a PPTM $\mathcal{A}$ that has access to the attacker tape ρ_{a} and t's interpretation, and attempts to compute t'. We must show that $\mathbb{P}_\rho \left[\mathcal{A}(1^\eta, \rho_a, [\![t]\!]^{\mathbb{M}}(1^\eta, \rho)) = [\![t']\!]^{\mathbb{M}}(1^\eta, \rho) \right] \in \mathsf{negl}(\eta)$.

The structure of the proof is the same as in Lemma 13. We first consider the evaluators $\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}$ and $\mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})}$, evaluating terms as specified in Definition 18, respectively in $(\Gamma; \mathcal{R})$ and $(\delta^a_c(\Gamma); \delta^a_c(\mathcal{R}))$. We consider an additional oracle $\mathcal{O}_1$, which is similar to $\mathcal{O}_0$, except that for any k_0, m_0, it answers encryption query $\mathcal{O}_1(m_0, \text{“}k\text{”}, \text{“}r\text{”})$ by checking that $k_0 = k$, and $m_0 = [\![m]\!]^{\mathbb{M}}(1^\eta, \rho)$, and returning the encryption of $[\![\mathsf{zeros}]\!]^{\mathbb{M}}(m_0)$ with k, r. All other queries (in particular for other random symbols r') are answered like $\mathcal{O}_0$. We establish that $\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}(t_0) = \mathcal{E}^{\mathcal{O}_2}_{\mathbb{M},\delta_c(\Gamma),\delta_c(\mathcal{R})}(\delta^a_c(t_0))$ for any t_0. That part of the proof is exactly as in Lemma 13, and we do not detail it again here.

As before, by Lemma 8 and Lemma 12 (which still holds), we have

$$\mathbb{P}_\rho \left[\mathcal{E}^{\mathcal{O}_0}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}}) = [\![t]\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}}) \right] \in \mathsf{ow}(\eta),$$

and

$$\mathbb{P}_\rho \left[\mathcal{E}^{\mathcal{O}_1}_{\mathbb{M},\Gamma,\mathcal{R}}(t)(1^\eta, \rho_{\mathsf{a}}) = [\![\delta^a_c(t)]\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}}) \right] \in \mathsf{ow}(\eta),$$

and similarly for t'.

We then construct an attacker $\mathcal{B}(1^\eta, \rho')$ playing the IND-CCA-2 game, depicted in Appendix A-A, where ρ' is w.l.o.g. seen as $(\rho'_{\mathsf{h}}, \rho_{\mathsf{a}})$, ρ'_{h} being the part of the random tape ρ_{h} not associated with k or r. The machine $\mathcal{B}$ internally simulates the execution of evaluator $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ (depending on the side β of the IND-CCA-2 game) on t and t', obtaining bitstrings m_β, m'_β. This way, $\mathcal{B}$ computes (with overwhelming probability):

- either $m_0 = [\![t]\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}})$ and $m'_0 = [\![t']\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}})$, when $\beta = 0$;
- or $m_1 = [\![\delta^a_c(t)]\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}})$ and $m'_1 = [\![\delta^a_c(t')]\!]^{\mathbb{M}}(1^\eta, \rho_{\mathsf{a}})$, when $\beta = 1$.

By applying $\mathcal{A}$ to m_β and checking whether it successfully computes m'_β, $\mathcal{B}$ guesses the value of β (0 if $\mathcal{A}$ succeeds, 1 otherwise). We can then show, just as in Lemma 13, that if $\mathcal{A}$ has a non-negligible probability of computing t' from t, then $\mathcal{B}$ has a non-negligible advantage in the IND-CCA-2 game. We do not repeat the argument here.

All that remains to be shown is how $\mathcal{B}$ is able to simulate the execution of $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ on t, t'. That proof is slightly more involved than in the symmetric case. As before, the main point is to simulate the oracle $\mathcal{O}_\beta$. We have shown how to do so for the symmetric oracles in $\mathcal{O}_\beta$, and need to extend that argument

$\underline{\mathsf{G}^{\mathsf{indcca},\beta}_{\mathcal{A}}(1^\eta,\rho)}$
$sk \leftarrow_\$ \{0,1\}^\eta$
$pk \leftarrow \mathsf{pk}(sk)$
$ch \leftarrow \bot$
$\beta' \leftarrow \mathcal{A}^{\mathcal{O}_\mathsf{d},\mathcal{O}_\mathsf{LR}}(pk, 1^\eta, \rho')$
return β'

$\underline{\mathcal{O}_\mathsf{d}(c)}$
if $ch = \bot \vee c \neq ch$ **then**
  return $[\![\mathsf{adec}]\!]^{\mathbb{M}}(1^\eta, c, sk)$
else
  return $\bot$

$\underline{\mathcal{O}_\mathsf{LR}(m_0,m_1)}$
$r \leftarrow_\$ \{0,1\}^\eta$
$ch \leftarrow [\![\mathsf{aenc}]\!]^{\mathbb{M}}(1^\eta, m_\beta, sk, r)$
return c

$\mathcal{A}$ can call $\mathcal{O}_\mathsf{LR}$ only once and $\mathcal{O}_\mathsf{d}$ any number of times. Note that $\mathcal{O}_\mathsf{d}$ answers any query before $\mathcal{O}_\mathsf{LR}$ has been called, and only queries other than the challenge ciphertext afterwards. ρ' is the tape ρ, except for the parts used by the left-right oracle and the key generation. The advantage of $\mathcal{A}$ is $\mathsf{Adv}^{\mathsf{indcca}}_{\mathcal{A}}(\eta) = \left|\mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcca},0}_{\mathcal{A}}(1^\eta,\rho)=1\right] - \mathbb{P}_\rho\left[\mathsf{G}^{\mathsf{indcca},1}_{\mathcal{A}}(1^\eta,\rho)=1\right]\right|$.

Fig. 7: The IND-CCA-2 game (in computational model $\mathbb{M}$)

to the asymmetric oracles $\mathcal{O}_{\mathsf{aenc}}$, $\mathcal{O}_{\mathsf{adec}}$, $\mathcal{O}_{\mathsf{pk}}$. $\mathcal{O}_{\mathsf{pk}}$ is easy, as $\mathcal{B}$ has access to $\mathsf{pk}(k)$. $\mathcal{O}_{\mathsf{aenc}}(m_0, \text{“}r_0\text{”}, \text{“}k_0\text{”})$, when $r_0 \neq r$, is similar: $\mathcal{B}$ has access to all $k_0 \neq k$, to $\mathsf{pk}(k)$, and to all $r_0 \neq r$, and can compute the encryption himself. When $r_0 = r$, $\mathcal{B}$ needs to call his encryption oracle (from the IND-CCA-2 game) to compute either the encryption of m_0 or of $[\![\mathsf{zeros}]\!]^{\mathbb{M}}(m_0)$, as expected.

The issue here is the unrestricted decryption oracle $\mathcal{O}_{\mathsf{adec}}$. When it is called on a key other than k, $\mathcal{B}$ can simply compute the decryption, as he knows the key. However, the case of $\mathcal{O}_{\mathsf{adec}}(c_0, \text{“}k\text{”})$ for an arbitrary c_0 is more subtle. $\mathcal{B}$ can only use the IND-CCA-2 decryption oracle before submitting the challenge to $\mathcal{O}_{\mathsf{LR}}$ – and he can only call it on $c_0 \neq [\![c]\!]^{\mathbb{M}}(1^\eta,\rho)$ afterwards. Therefore, we must ensure that $\mathcal{B}$ never needs to make a forbidden decryption query after submitting the challenge. That situation could possibly arise, *e.g.* if $\mathcal{B}$ started to evaluate a subterm of t that contains c, requiring a call to $\mathcal{O}_{\mathsf{LR}}$, before evaluating another subterm where another term c' needs decryption, that happens to have the same interpretation as c.

To get around this issue, $\mathcal{B}$ evaluates the terms t, t' in a slightly different order than $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$. Note that, since $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ is not stateful, any order in which it evaluates terms leads to the same result.

$\mathcal{B}$ goes recursively through the terms as $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ would, until it reaches an occurrence of subterm c, the ciphertext to be replaced. He then pauses the evaluation for that branch of the term, and continues to evaluate the rest of t and t'. While doing so, $\mathcal{B}$ never calls $\mathcal{O}_{\mathsf{LR}}$, as he never needs to evaluate c. Thus, any decryption is still permitted. Then, $\mathcal{B}$ evaluates m, the challenge plaintext. Since c cannot be a subterm of m (as it is itself c's subterm), $\mathcal{B}$ can evaluate m completely. Next, $\mathcal{B}$ calls $\mathcal{O}_{\mathsf{LR}}$ on m and the corresponding null bitstring, to evaluate c (or the corresponding encryption of 0s, on the right). He then finishes the evaluation of t and t', *i.e.* goes back up through the terms, from all positions of subterm c in t, t'. While doing so, he may encounter a subterm $\mathsf{adec}^\star(c', k, \langle c_i\rangle_i)$, for some c' which admits c as a subterm. That decryption must be evaluated using oracle $\mathcal{O}_{\mathsf{adec}}$. We thus need to ensure $[\![c']\!]^{\mathbb{M}}(1^\eta,\rho) \neq [\![c]\!]^{\mathbb{M}}(1^\eta,\rho)$. By construction of $\mathcal{E}^{\mathcal{O}_\beta}_{\mathbb{M},\Gamma,\mathcal{R}}$ and typing rule ADEC*, we know that $[\![c']\!]^{\mathbb{M}}(1^\eta,\rho) \neq [\![c_i]\!]^{\mathbb{M}}(1^\eta,\rho)$ for all i, and that $\langle c_i\rangle_i$ contains all subterms of c' that are ciphertexts – in particular c. Thus, c' and c have different interpretations, and the decryption is permitted. This allows $\mathcal{B}$ to finish simulating the evaluation of t, t', which concludes the proof. □

Similarly to the symmetric case, we then define δ^a_Γ, the operation that adds the symbol zeros in all ciphertexts in a given term t, *i.e.* the iteration of δ^a_c for all ciphertexts c in t that use a key in Γ. We apply Lemma 22 repeatedly, to show that, informally, $\delta^a_\Gamma(t) \blacktriangleright \delta^a_\Gamma(t')$ implies $t \blacktriangleright t'$ for any t, t' (similarly to Lemma 15).

Finally, the statement of Theorem 2 is unchanged. The updates to its proof are straightforward: in the case of rule NAME, we use the previous property to add zeros in all asymmetric ciphertexts, as we did with Lemma 15 for symmetric ciphertexts, and the rest of the proof remains the same.

C. Elimination of adec

In Appendix D, we have proved that if a term t type-checks in the system for base terms, then there exists a term $t' \approx t$ that type-checks in the restricted system. We extend that result to the case of asymmetric encryption. To do so, we remove from t all subterms typed with the rule ADEC of the base system, and replace them with terms containing $\mathsf{adec}^\star$, that can be typed with rule ADEC*. The idea is the same as when removing occurrences of SDEC using the INT-CTXT assumption. In the asymmetric case however, we could not remove decryption entirely, and have handled that difficulty in the previous subsection using $\mathsf{adec}^\star$ and IND-CCA-2. The following replacement lemma, analogous to Lemma 16 is thus simpler than in the symmetric case.

Lemma 23. *Let* $(\Gamma;\mathcal{R})$ *be a well-formed mapping environment,* T *a message type,* k *a key with* $\Gamma(k) =$ AK[T]*, and* t *a ground term such that* $\Gamma;\mathcal{R} \vdash t :$ Msg*. Let* $\mathsf{aenc}(p_1,\mathsf{pk}(k),r_1),\ldots,\mathsf{aenc}(p_\ell,\mathsf{pk}(k),r_\ell)$ *be the sequence of encryptions using public key* $\mathsf{pk}(k)$ *and randoms in* $\mathcal{R}^{\mathcal{B}}$ *that occur in* t*. Let*

$$\delta^a(t,k) = \begin{cases} \mathsf{if}\ t = \mathsf{aenc}(p_1,\mathsf{pk}(k),r_1)\ \mathsf{then}\ p_1\ \mathsf{else} \\ \quad \ldots \\ \quad \mathsf{if}\ t = \mathsf{aenc}(p_\ell,\mathsf{pk}(k),r_\ell)\ \mathsf{then}\ p_\ell \\ \quad \mathsf{else}\ \mathsf{adec}^\star(t,k,\langle \mathsf{aenc}(p_i,\mathsf{pk}(k),r_i)\rangle_{1\leq i\leq \ell}) \end{cases}$$

We have $\mathsf{adec}(t,k) \approx \delta^a(t,k)$.

Proof. If $[\![t]\!]^{\mathbb{M}}(1^\eta,\rho) = [\![\mathsf{aenc}(p_i,\mathsf{pk}(k),r_i)]\!]^{\mathbb{M}}(1^\eta,\rho)$ for some i, then $\delta^a(t,k)$ correctly evaluates to $[\![p_i]\!]^{\mathbb{M}}(1^\eta,\rho)$. Otherwise, $\delta^a(t,k)$ evaluates to $[\![\mathsf{adec}^\star(t,k,\langle\mathsf{aenc}(p_i,\mathsf{pk}(k),r_i)\rangle_{1\le i\le \ell})]\!]^{\mathbb{M}}(1^\eta,\rho)$, in the innermost "else" branch. That term, by definition, has the same semantics as $\mathsf{adec}(t,k)$, since $[\![t]\!]^{\mathbb{M}}(1^\eta,\rho) \neq [\![\mathsf{aenc}(t_i,\mathsf{pk}(k),r_i)]\!]^{\mathbb{M}}(1^\eta,\rho)$ for any i, which concludes the proof. □

Finally, in the proof of Lemma 17 used to established Theorem 3, we use Lemma 23 for the case of rule ADEC, in the same manner as Lemma 16 was used for the case of rule SDEC.

D. Some case studies

We apply our framework on two protocols relying on asymmetric encryption.

Needham-Schroeder-Lowe protocol: We consider the tagged version of this protocol informally described below [25].

$$\begin{array}{ll} A \to B: & \{1, N_a, A\}_{\mathsf{pk}(B)} \\ B \to A: & \{2, N_a, N_b, B\}_{\mathsf{pk}(A)} \\ A \to B: & \{3, N_b\}_{\mathsf{pk}(A)} \end{array}$$

We consider secrecy of the nonce N_b from the point of view of each role and we consider that some agents may be corrupted. In order to type the protocol, we give type $\mathrm{AK}[\mathrm{T}_0 + \mathrm{Low}]$ to $\mathsf{ska}[i]$ – the symbol used to represent private keys of honest agents $\mathsf{a}[i]$ – with T_0 as follows:

$$\begin{array}{ll} & \mathrm{Cst}_1^0 \times \mathrm{High} \times \mathrm{Cst}_{\mathsf{a}}^\infty \\ + & \mathrm{Cst}_2^0 \times \mathrm{Msg} \times \mathrm{High} \times \mathrm{Cst}_{\mathsf{a}}^\infty \\ + & \mathrm{Cst}_3^0 \times \mathrm{High} \end{array}$$

The modelling of this protocol is made up of two actions per role, and we have to consider two cases: the honest agent executing the role can be engaged in the communication with an honest agent or a dishonest one. Therefore, in total, we have 8 actions to consider. We detail below the typing of the second action of the role A when the agent is engaged in the communication with a dishonest responder. In this case, the term outputted by $\mathsf{a}[i]$ responding to $\mathsf{a}^{\mathsf{d}}[j]$ in session k is of the form:

$$\mathsf{aenc}(\langle 3, \mathsf{fst}(\mathsf{snd}(\mathsf{snd}(t^{\mathsf{d}}_{\mathsf{adec}}))))\rangle, \mathsf{pk}(\mathsf{ska}^{\mathsf{d}}[j]), \mathsf{r}^{\mathsf{d}}_{\mathsf{a}}[i,j,k])$$

where $t^{\mathsf{d}}_{\mathsf{adec}} = \mathsf{adec}(\mathsf{input}@A^{\mathsf{d}}_1[i,j,k], \mathsf{ska}[i])$, and $\mathsf{r}^{\mathsf{d}}_{\mathsf{a}}$ is a name having Low type (used here as a random). However, this output is performed under some conditions, in particular, $\mathsf{a}[i]$ will test the tag as well as that $\mathsf{snd}(\mathsf{snd}(\mathsf{snd}(t_{\mathsf{adec}}))) = \mathsf{a}^{\mathsf{d}}[j]$. When typing this output, we will have to consider the case where $t^{\mathsf{d}}_{\mathsf{adec}}$ is of type T_0, and the case where $t^{\mathsf{d}}_{\mathsf{adec}}$ is of type Low. The latter one will be easily handled using the rule FUN-LOW. To handle the former one, we will rely on the tag, and also on the test done on the agent name to show that actually the condition will always fail, and thus the resulting output will be the constant empty of type Low. This additional check on the name of the responder is crucial to show that the protocol is well-typed and explain also why the secrecy of the nonce N_b can not be established on the well-known flawed Needham-Schroeder protocol using our technique.

To establish the secrecy of N_b from the point of view of A, we have to show how to type

$$\mathsf{if}\ \Phi\ \mathsf{then}\ \mathsf{fst}(\mathsf{snd}(\mathsf{snd}(t_{\mathsf{adec}})))\ \mathsf{else}\ n_{\mathsf{fresh}}$$

where $t_{\mathsf{adec}} = \mathsf{adec}(\mathsf{input}@\mathsf{A}_1[i,j,k], \mathsf{ska}[i])$, $\mathsf{n}_{\mathsf{fresh}}$ is a name having type High, and Φ is a conjunction of conditions expressing in particular the check regarding the tag, and the check performed by A on its own nonce. This last check is actually crucial to deal with, relying on the rule EQ-FALSE, the case where t_{adec} will be assumed of type Low. The nonce N_a generated by A can not be equal to something having type Low, and thus we know that the condition will fail, and we only have to type $\mathsf{n}_{\mathsf{fresh}}$ in this case.

Mechanism 6 (ISO-11770/part III): This protocol is informally described below. The norm does not mention any tagging mechanism, and we do not have to assume one for applying our typing result.

$$\begin{array}{ll} A \to B: & \{A, K_A, R_A\}_{\mathsf{pk}(B)} \\ B \to A: & \{B, K_B, R_A, R_B\}_{\mathsf{pk}(A)} \\ A \to B: & R_B \end{array}$$

We consider here secrecy of the key K_B from the point of view of each role. Actually, the situation is rather similar to the case of the Needham-Schroeder-Lowe protocol, and we are able to type the protocol, as well as the terms representing the secrecy properties of the key K_B. It can be done using type $\mathrm{AK}[\mathrm{Cst}_{\mathsf{a}}^\infty \times \mathrm{High} \times \mathrm{Msg} + \mathrm{Low}]$ for typing ska.