Secrecy and Authenticity Types for Secure Distributed Messaging*

Michele Bugliesi, Stefano Calzavara, and Damiano Macedonio

Università Ca' Foscari Venezia, Dipartimento di Informatica
{michele,scalzava,mace}@dsi.unive.it

Abstract. We introduce a calculus with mobile names, distributed principals and primitives for secure remote communication, without any reference to explicit cryptography. The calculus is equipped with a system of types and effects providing static guarantees of secrecy and authenticity in the presence of a Dolev-Yao intruder. The novelty with respect to existing type systems for security is in the structure of our secrecy and authenticity types, which are inspired by the formulas of BAN Logic, and retain much of the simplicity and intuitive reading of such formulas. Drawing on these types, the type system makes it possible to characterize authenticity directly as a property of the data exchanged during a protocol rather than indirectly by extracting and interpreting the effects the protocol has on that data.

1 Introduction

Distributed protocols draw on cryptographic constructs to protect the secrecy and integrity of sensitive data against any potential attack. When designing distributed applications, however, it is often convenient to rely on more abstract, structured primitives for secure remote messaging and let a compiler automatically build defensive implementations on top of the underlying cryptographic infrastructure.

Following an increasingly popular approach in the specification of distributed systems, in [7, 6] the first author and Focardi isolated a core set of security abstractions for programming distributed protocols, and showed them effective both for high-level protocol design and for security analysis in adversarial settings. In the present paper, we further investigate the effectiveness of that approach by developing a typed version of the abstractions.

We introduce a variant of the calculus in [6], that features mobile names, distributed principals and primitives for secure remote communication, without any reference to explicit cryptography. The calculus is equipped with a system of types and effects which disciplines the use of the messaging abstractions to provide static guarantees of strong secrecy and strong authenticity in the presence of a Dolev-Yao intruder. Strong secrecy is formalized in terms of behavioral

* Work partially supported by MIUR Projects SOFT "*Security Oriented Formal Techniques*" and IPODS "*Interacting Processes in Open-ended Distributed Systems*".

equivalence in the style of Abadi's seminal work [1]. Authenticity, in turn, is proved by establishing a form of injective agreement between statements and expectations in the style of [13].

The novelty with respect to existing security type systems is in the choice of our secrecy and authenticity types. Specifically, the type system makes it possible to characterize authenticity directly as a property of the data exchanged during a protocol rather than indirectly by extracting and interpreting the effects the protocol has on that data. Our types are inspired by the formulas of BAN Logic [8], and retain much of the simplicity and intuitive reading of such formulas; their use for type-checking similarly inherits the declarative style of BAN Logic's deductive system. The simple structure of the types, and the relative ease of type-checking also derive from the high-level nature of the underlying process calculus and its messaging primitives, that encapsulate and abstract away all cryptographic details. To illustrate, we may write a principal specification of the form[1]:

$$\mathsf{receive}\ (\mathsf{x} : \mathsf{From}(\mathsf{p}))\ \mathsf{on}\ \mathsf{a}.P$$

to mean that we expect a piece of data on channel a from principal p, and be guaranteed statically that, in all protocol runs to which this principal participates, any piece of data received on a at this step does indeed come from p.

In spite of their simplicity, the calculus and the type system are rather expressive and support a wide range of distributed protocol idioms. We exemplify the practical effectiveness of our approach by showing the type system at work on the security analysis of a variant of the 2KP e-payment protocol [5].

Plan. §2 presents the calculus, §3 the system of types and effects. §4 details the secrecy and authenticity properties enforced by the system. §5 presents the e-payment protocol case study, and §6 concludes the presentation. We omit proofs for lack of space (see [9] for details).

2 The Calculus

Syntax. We presuppose three countable sets of disjoint symbols, for principal identities p, q, r, (channel) names a, b, c and variables w, x, y, z. We let m, n range over all names and u, v over names or variables when the distinctions are immaterial. Tuples are indicated by a tilde, as in $\tilde{n}, \tilde{x}, \tilde{v}$.

The calculus is a variant of the calculus from [7], where we structure the syntax in two layers, for networks and processes, and introduce the explicit network form $p[\![P]\!]$ to note the process P running on behalf of the principal identity p.

$$\begin{array}{llcll} \textit{Networks} & M, N, O & ::= & p[\![P]\!] & \text{(Principal)} \\ & & \mid & M|N & \text{(Parallel)} \\ & & \mid & (\nu a : \tau)M & \text{(Restriction)} \\ & & \mid & \mathbf{0} & \text{(Empty)} \end{array}$$

[1] The actual syntax of our calculus is different from the one used in this illustrative example.

Processes	P, Q, R	$::= \overline{u}@v\langle \underline{*} : \tilde{u}\rangle^{\circ}$	(Output)
		$\mid\ a(\underline{u} : \tilde{x})^{\circ}.P$	(Input)
		$\mid\ 0$	(Inaction)
		$\mid\ P \mid Q$	(Parallel)
		$\mid\ \text{test}(u = v) \text{ then } P \text{ else } Q$	(Choice, test $\in$ {if,check})
		$\mid\ \text{rec } X.P$	(Recursion)
		$\mid\ X$	(Recursion Var)
		$\mid\ (\nu a : \tau)P$	(Restriction)

The null, parallel composition, and recursive forms are just as in the pi-calculus. Conditionals are also like pi-calculus matching: for typing purposes, however, we use a special syntax to express the matching that corresponds to nonce-checks. The restriction operators (both for processes and networks) have the familiar pi-calculus syntax, but weaker scoping rules (see below); they are annotated with tagged types to be described in § 3. As to input/output, we have various messaging forms, depending on the instantiation of $\underline{*}$, $\underline{u}$ and $\circ$. The notation $\underline{*}$ stands for $*$ or '$-$' ($\underline{u}$ similarly abbreviates u or '$-$'), where '$-$' can be interpreted as an *anonymous* identity; instead, $\circ$ is short for $\bullet$ or the empty string ε. The intuitive reading is as follows: $\overline{u}@v\langle \underline{*} : \tilde{u}\rangle^{\circ}$ denotes an output on channel u directed to principal v with payload $\tilde{u}$; the payload is certified as originating from the sender if $\underline{*}$ is $*$, and it is secret for the receiver v if $\circ$ is $\bullet$. Dually, $a(\underline{u} : \tilde{x})^{\circ}.P$ denotes an input on a from principal $\underline{u}$ of the payload $\tilde{x}$. As we detail below, input and output must agree on the secrecy and authenticity tags to synchronize. Note, finally, that the subject of an input must be a channel (not a variable): like in the *local* pi-calculus [18], we thus disallow the transmission of the input capability on any channel. The notions of free and bound names and variables arise as expected.

Semantics. We formalize the semantics of networks via a *labelled transition system*. Following [2] (and the MIM semantics of [6]) in our transition system two principals can never synchronize directly; rather, every interaction requires the mediation of the intruder, which intercepts each message exchaged and then delivers it to the recipient whe the recipient is ready to consume it. To formalize the dynamics of message interception, we extend the syntax of networks with an additional, run-time process form to represent the copies of the messages stored (or cached) upon interception:

$$M, N ::= \ldots \text{as above} \ldots \mid \overline{c}@q\langle \underline{p} : \tilde{m} \| \tilde{n}\rangle^{\circ}_{i}$$

Each intercepted output is cached at a fresh index i which is generated upon interception and remains available as a reference to the cached output. The cached output exhibits two views of the message content: the actual payload $\tilde{m}$, and the view of the payload $\tilde{n}$ as available to an external observer. As we discuss below, the two views differ when the intercepted output bears the secrecy tag $\circ = \bullet$.

Table 1 Labelled Transition System

(Input)
$$\frac{\sigma = \{\tilde{m}/\tilde{x}\}}{p[\![c(\underline{q} : \tilde{x})^{\circ}.P]\!] \xrightarrow{c@p(\underline{q}:\tilde{m})^{\circ}} p[\![P\sigma]\!]}$$

(Plain Output)
$$\frac{p \neq q, \quad \underline{p} = p \text{ iff } \underline{*} = *, \quad i \text{ fresh}}{p[\![\overline{c}@q\langle \underline{*} : \tilde{m}\rangle]\!] \xrightarrow{(i)\overline{c}@q\langle \underline{p}:\tilde{m}\rangle_i} \overline{c}@q\langle \underline{p} : \tilde{m} \| \tilde{m}\rangle_i}$$

(Secret Output)
$$\frac{p \neq q, \quad \underline{p} = p \text{ iff } \underline{*} = *, \quad |\tilde{n}| = |\tilde{m}|, \quad i, \tilde{n} \text{ fresh}}{p[\![\overline{c}@q\langle \underline{*} : \tilde{m}\rangle^{\bullet}]\!] \xrightarrow{(i,\tilde{n}:Public)\overline{c}@q\langle \underline{p}:\tilde{n}\rangle_i^{\bullet}} \overline{c}@q\langle \underline{p} : \tilde{m} \| \tilde{n}\rangle_i^{\bullet}}$$

(Open Process)
$$\frac{(\nu a : \tau)p[\![P]\!] \xrightarrow{\alpha} N}{p[\![(\nu a : \tau)P]\!] \xrightarrow{\alpha} N}$$

(Open Network)
$$\frac{N \xrightarrow{(i,\tilde{b}:\tilde{\tau})\overline{c}@q\langle \underline{p}:\tilde{m}\rangle_i^{\circ}} N', \ a \in \{\tilde{m}, c\} \setminus \{\tilde{b}, i\}}{(\nu a : \tau)N \xrightarrow{(i,\tilde{b}:\tilde{\tau},a:\tau)\overline{c}@q\langle \underline{p}:\tilde{m}\rangle_i^{\circ}} N'}$$

(Forward)
$$\frac{N \equiv (\nu\tilde{a} : \tilde{\tau})(\hat{N}|\overline{c}@q\langle p : \tilde{m}\|\tilde{n}\rangle_i^{\circ}), \quad \hat{N} \xrightarrow{c@q(p:\tilde{m})^{\circ}} N'}{N \xrightarrow{(i)} (\nu\tilde{a} : \tilde{\tau})N'}$$

(Replay)
$$\frac{N \equiv (\nu\tilde{a} : \tilde{\tau})(\hat{N}|\overline{c}@q\langle - : \tilde{m}\|\tilde{n}\rangle_i^{\circ}), \quad \hat{N} \xrightarrow{c@q(-:\tilde{m})^{\circ}} N'}{N \xrightarrow{(i)} (\nu\tilde{a} : \tilde{\tau})(N'|\overline{c}@q\langle - : \tilde{m}\|\tilde{n}\rangle_i^{\circ})}$$

(Test True)
$$\frac{p[\![P]\!] \xrightarrow{\alpha} p[\![R]\!]}{p[\![\text{test } (m = m) \text{ then } P \text{ else } Q]\!] \xrightarrow{\alpha} p[\![R]\!]}$$

(Test False)
$$\frac{p[\![Q]\!] \xrightarrow{\alpha} p[\![R]\!], \quad m \neq n}{p[\![\text{test } (m = n) \text{ then } P \text{ else } Q]\!] \xrightarrow{\alpha} p[\![R]\!]}$$

(Recursion)
$$\frac{p[\![P\{\text{rec } X.P/X\}]\!] \xrightarrow{\alpha} p[\![P']\!]}{p[\![\text{rec } X.P]\!] \xrightarrow{\alpha} p[\![P']\!]}$$

(New)
$$\frac{N \xrightarrow{\alpha} N', \quad a \notin n(\alpha)}{(\nu a : \tau)N \xrightarrow{\alpha} (\nu a : \tau)N'}$$

(Parallel Process)
$$\frac{p[\![P]\!]|p[\![Q]\!] \xrightarrow{\alpha} N}{p[\![P|Q]\!] \xrightarrow{\alpha} N}$$

(Parallel Network)
$$\frac{M \xrightarrow{\alpha} M', \quad bn(\alpha) \cap fn(N) = \emptyset}{M|N, N|M \xrightarrow{\alpha} M'|N, N|M'}$$

The labelled transitions are collected in Table 1. We comment on the input/output and replay/forward rules, the remaining rules are standard. The (Input) rule allows a principal to input values from the network and proceed after propagating the bindings for the input variables to the continuation process. The (Output) rules formalize the interplay between output and interception. An output generates a label and caches a copy of the intercepted message at a fresh index. The label shows the view of the output available to an observer, while the cached copy holds both the internal and the external view of the payload. If the output is plain, the two views coincide; if it is secret, the external view is a tuple $\tilde{n}$ of fresh names: as discussed in [7], this corresponds to assume an implementation of a secret output in terms of randomized encryption. The (Open) rules implement the scope extrusion mechanisms: notice that channel names are always extruded in an output. All bound names (bar the intercept indexes) in the output labels come with associated type annotations: these bear no computational/observational meaning, and are only convenient when we formalize the properties of the type system. The (Replay) and (Forward) rules allow two principal to synchronize via the intruder by transmitting the cached outputs to their intended receivers. If the original output was certified, the cached copy gets consumed when used; non-certified outputs, instead, may be replayed back to an input-ready principal any number of times. It is worth noticing that private (restricted) names exchanged via a secret output are never exposed to an observer, unless of course the receiver leaks them. To see that, notice that the exchange arises as a result of a (Secret Output) transition, which does not expose the private names of its payload, followed by a (Replay)/(Forward) transition, which only exhibits the index of the intercepted message. The secret exchange of restricted names is still possible, and achieved in (Forward) and (Reply), relying on structural congruence for scope extrusion. The definition of structural congruence is standard, and omitted for brevity.

3 The Type System

Types, effects and type environments. The type system is built around types (T, U, V) and effects (E, F), both assigned to values and variables. The structure and intuitive reading of types is as follows:

- *Public*: values that are known or can be leaked publicly;
- *Secret*$(\tilde{u})$: values that must be kept secret among the principals in $\tilde{u}$;
- *Any*: values with unknown status, might be either *Public* or *Secret*;
- *Chan*$\langle\tilde{T};\tilde{U}\rangle$: channels with payload a tuple of type $\tilde{T}, \tilde{U}$;
- *Prin*: principal identities.

When occurring in a type (and later in an effect), the notation $\tilde{u}$ indicates a set rather than a tuple. In a channel type $Chan\langle\tilde{T};\tilde{U}\rangle$, we assume that each $U \in \tilde{U}$ is a *Secret* type, while each $T \in \tilde{T}$ is a type other than *Secret*. *Secret* types allow us to define *groups of secrecy* (much in the spirit of the work in [10]) and these, in turn, will be instrumental in deriving authenticity judgements. *Any* plays the

Table 2 Types and environments formation

$$\frac{\Gamma;\Delta \vdash \diamond, \quad \tau \textit{ consistent}, \quad \mathit{fn}(\tau) \cup \mathit{fv}(\tau) \subseteq \mathit{dom}(\Gamma) \cup \mathit{dom}(\Delta)}{\Gamma;\Delta \vdash \tau} \text{ (Good Type)}$$

$$\frac{}{\emptyset;\emptyset \vdash \diamond} \text{ (Empty)} \qquad \frac{\Gamma;\Delta \vdash T, \quad u \notin \mathit{dom}(\Gamma)}{\Gamma, u : T;\Delta \vdash \diamond} \text{ (Type)} \qquad \frac{\Gamma;\Delta \vdash \tilde{E}, \quad u \notin \mathit{dom}(\Delta)}{\Gamma;\Delta, u : \tilde{E} \vdash \diamond} \text{ (Effect)}$$

same role as in [1]: values with this type must be protected as secrets, but cannot be used as secrets, because they might in fact be public.

As to effects, their purpose is to encode time-dependent information about values: as such, unlike types, they are not invariant through reduction. Their syntax and intuitive reading is as follows:

- $\mathit{From}^p(\tilde{u})$: values received by p coming from any of the principals $\tilde{u}$;
- $\mathit{Fresh}^p(\tilde{u})$: fresh values received by p coming from any of the principals $\tilde{u}$;
- $\mathit{With}^p(\tilde{u})$: values received by p in a message with payload $\tilde{u}$;
- $\mathit{Nonce}(p)$ and $\mathit{Checked}(p)$: nonces created/checked[2] by p.

The effects *From*, *Fresh* and *With* are only associated with variables (not names or identities), as they express properties that pertain to the transmission of a name (hence to the variable where the name gets received) rather than to the name itself. The effects *Nonce* and *Checked*, in turn, help to single out the steps of nonce verification.

A type T and a set of effects $\tilde{E}$ can be composed to form what we call a *tagged type*, noted $T/\tilde{E}$. We let τ range over tagged types, and use T and $T/\emptyset$ interchangeably, as we do for $\tilde{E}$ and $\mathit{Any}/\tilde{E}$. Also, we let $\tau.T$ and $\tau.E$ denote the type and effect components of a tagged type, respectively. Throughout, we assume that the effect sets in a tagged type are *consistent*, that is they do not contain more that one Nonce and/or $\mathit{Checked}$ element. Further, we introduce the following notion of *generative* types, i.e., of types that may legally be associated with fresh names: $T/\tilde{E}$ is generative for a principal p iff $T \in \{\mathit{Public}, \mathit{Secret}(\tilde{u}), \mathit{Chan}\langle\tilde{U};\tilde{V}\rangle\}$ and $\tilde{E} \subseteq \{\mathit{Nonce}(p)\}$; τ is generative for a network M iff $M \equiv (\nu\tilde{a} : \tilde{\tau})(p[\![P]\!] \mid M')$ and τ is generative for p.

The type system derives various forms of typing and subtyping judgements. The type environments for these judgements consist of two components, Γ and Δ, mapping names and variables to types and (sets of) effects, respectively. All environments occurring in a typing judgement must be well-formed, according to the rules in Table 2. We make the implicit assumption that the effects associated with a variable do not include Nonce elements, and dually, that the effects given

[2] Although, strictly speaking, *Nonce* and *Checked* are not time-dependent, it is technically convenient to treat them as effects.

Table 3 Ordering on types and effects

(IdPublic)	(ChanPublic)	(FreshFrom)
$\overline{Prin \leq Public}$	$\overline{Chan\langle \tilde{T}; \tilde{U} \rangle \leq Public}$	$\overline{Fresh^p(\tilde{v}) \leq From^p(\tilde{v})}$
(NonceChecked)	(ContraWith) $\tilde{u} \subseteq \tilde{v}$	
$\overline{Nonce(p) \leq Checked(p)}$	$\overline{With^p(\tilde{v}) \leq With^p(\tilde{u})}$	
(CoFresh) $\tilde{v} \subseteq \tilde{u}$	(CoFrom) $\tilde{v} \subseteq \tilde{u}$	(EffectSet) $\forall F{\in}\tilde{F}.\ \exists E{\in}\tilde{E}.\ E \leq F$
$\overline{Fresh^p(\tilde{v}) \leq Fresh^p(\tilde{u})}$	$\overline{From^p(\tilde{v}) \leq From^p(\tilde{u})}$	$\overline{\tilde{E} \leq \tilde{F}}$

to names only include $Nonce$ and $Checked$ elements. We use the notation $\tilde{v} : \tilde{\tau}$ for $v_1 : \tau_1, \ldots, v_n : \tau_n$ and $\tilde{v} : \tau$ for $v_1 : \tau, \ldots, v_n : \tau$. In addition, we introduce the following notation to single out various components of a type environment:

$$Nonces(\Gamma; \Delta) = \{n \mid \Gamma; \Delta \vdash n : Nonce(p)\}$$
$$Secrets(\Gamma; \Delta) = \{a \mid \Gamma; \Delta \vdash a : Secret(\tilde{u})\}.$$

Ordering on types and effects. Types and effects are organized in the pre-order relation defined by the rules in Table 3. Rules (IdPublic) and (ChanPublic) imply that trusted identities are public, and so are channel names, which indeed are always leaked upon output. Rule (FreshFrom) states that $Fresh^p(\tilde{v})$ has stronger authenticity guarantees than $From^p(\tilde{v})$. Rule (NonceChecked) states that a new nonce can safely be promoted to the status "checked": this is sound, as in our our system the effect $Nonce$ may only be assigned to names, not to variables. The pre-order on effects is lifted to sets of effects using the standard *upper* powerset (or *Egli-Milner*) construction: the resulting relation is still a pre-order with set union as (total) meet operator. The remaining rules in the table are self-explanatory. In addition, we define Any as the top element of the pre-order (on types) and stipulate that $T/\tilde{E} \leq T'/\tilde{E}'$ iff $T \leq T'$ and $\tilde{E} \leq \tilde{E}'$. Finally, we let $\tau = \tau'$ if and only if $\tau \leq \tau'$ and $\tau' \leq \tau$.

Typing of values. The typing rules for values are collected in Table 4. The (Domain) and (Subsumption) rules are standard, and (Tag) is self-explanatory. The two correlation rules are inspired by BAN Logic [8]: (Correlation Fresh) states that if y is a nonce checked by p, then any name received by p with y must be fresh; (Correlation From) states that if y is a shared secret between p and $\tilde{u}$, then any name received by p with y must come from a principal in $\tilde{u}$. The (Combine) rule is used to gather the information inferred by the correlation rules.

Table 4 Typing of values

(Domain)
$$\frac{\Gamma;\Delta \vdash \diamond, \quad \Gamma(u) = \tau \vee \Delta(u) = \tau}{\Gamma;\Delta \vdash u : \tau}$$

(Subsumption)
$$\frac{\Gamma;\Delta \vdash u : \tau, \quad \tau \leq \tau', \quad \Gamma;\Delta \vdash \tau'}{\Gamma;\Delta \vdash u : \tau'}$$

(Tag)
$$\frac{\Gamma;\emptyset \vdash u : T, \quad \emptyset;\Delta \vdash u : \tilde{E}}{\Gamma;\Delta \vdash u : T/\tilde{E}}$$

(Correlation Fresh)
$$\frac{\Gamma;\Delta \vdash y : Checked(p), \quad \Gamma;\Delta \vdash x : With^p(y)}{\Gamma;\Delta \vdash x : Fresh^p}$$

(Combine)
$$\frac{\Gamma;\Delta \vdash x : Fresh^p, \quad \Gamma;\Delta \vdash x : From^p(\tilde{u})}{\Gamma;\Delta \vdash x : Fresh^p(\tilde{u})}$$

(Correlation From)
$$\frac{\Gamma;\Delta \vdash y : Secret(p,\tilde{u}), \quad \Gamma;\Delta \vdash x : With^p(y)}{\Gamma;\Delta \vdash x : From^p(\tilde{u})}$$

Typing of processes. Processes are always type-checked with respect to a principal identity. The typing judgement for processes has the form $\Gamma;\Delta \vdash_p P$, where p is intended to be the identity of the principal running P. We start illustrating the typing of processes with the rules for input/output in Table 5.

Rule (Public Output) governs the communication of public values, which is legal provided that the type of the payload is consistent with the type expected by the channel. Note that the output can be either plain or secret. The (Secret Output) rule requires both the sender and the receiver to be part of the secrecy group declared for each secret included in the message.

As for input, in all rules the continuation process is type-checked against an environment that stores the interdependence of each input variable with all the remaining components of the message received. If the sender is unknown, the payload type declared by the channel is ignored, as the message may come from the intruder. In absence of adequate guarantees about the sender, the input variables associated to public positions can be safely treated as public, while those associated to secret positions must be given type *Any* (as they might be secret, if the sender is well-typed, or anything when the sender is untyped). On the other hand, if the sender of a message is a known principal, the receiver can trust the payload type of the channel.

The last two rules in Table 5 involve the conditional forms. Following [15], the (If) rule exploits the equality between u and v to refine the types of the two values in the typing for the *then* branch. The notation $\Gamma \sqcap u : V$ indicates the environment $\Gamma, u : V$ if $u \notin dom(\Gamma)$, otherwise the environment $\Gamma \setminus \{u : U\}, u : U \sqcap V$, where $\sqcap$ is the partial meet operator on types. Notice that we only refine the types of u and v and not their effects, as refining the effects would be unsound. To illustrate, given the assumption $u : From^p(q)$, associating the same effect to v would be unsound, as v might come from another principal r, even though $u = v$. The (Check) rule implements a nonce verification mechanism, and

Table 5 Typing of processes: input/output and conditionals

(Public Output)

$$\frac{\Gamma;\Delta \vdash u : Chan\langle \tilde{T};\emptyset\rangle, \quad \Gamma;\Delta \vdash \tilde{u} : \tilde{T}, \quad \Gamma;\Delta \vdash v : Prin}{\Gamma;\Delta \vdash_p \overline{u}@v\langle \underline{*} : \tilde{u}\rangle^{\circ}}$$

(Secret Output)

$$\frac{\begin{array}{ll}\Gamma;\Delta \vdash u : Chan\langle \tilde{T};\tilde{U}\rangle, & \forall i\,(U_i = Secret(\tilde{v}_i) \wedge p, v \in \tilde{v}_i),\\ \Gamma;\Delta \vdash \tilde{u} : (\tilde{T},\tilde{U}), & \Gamma;\Delta \vdash v : Prin\end{array}}{\Gamma;\Delta \vdash_p \overline{u}@v\langle \underline{*} : \tilde{u}\rangle^{\bullet}}$$

(Non-certified Input)

$$\frac{\begin{array}{l}\Gamma;\Delta \vdash a : Chan\langle \tilde{T};\tilde{U}\rangle, \quad |\tilde{x}| = |\tilde{T}| \wedge |\tilde{y}| = |\tilde{U}|,\\ \Gamma,\tilde{x} : Public, \tilde{y} : Any; \Delta, \tilde{x} : With^p(\tilde{x},\tilde{y}), \tilde{y} : With^p(\tilde{x},\tilde{y}) \vdash_p P\end{array}}{\Gamma;\Delta \vdash_p a(- : \tilde{x},\tilde{y})^{\circ}.P}$$

(Trusted Input)

$$\frac{\begin{array}{l}\Gamma;\Delta \vdash a : Chan\langle \tilde{T};\tilde{U}\rangle, \quad \Gamma;\Delta \vdash u : Prin, \quad |\tilde{x}| = |\tilde{T}| \wedge |\tilde{y}| = |\tilde{U}|,\\ \Gamma,\tilde{x} : \tilde{T}, \tilde{y} : \tilde{U};\ \Delta, \tilde{x} : \{Fresh^p(u), With^p(\tilde{x},\tilde{y})\}, \tilde{y} : \{Fresh^p(u), With^p(\tilde{x},\tilde{y})\} \vdash_p P\end{array}}{\Gamma;\Delta \vdash_p a(u : \tilde{x},\tilde{y})^{\circ}.P}$$

(If)

$$\frac{\Gamma;\emptyset \vdash u : U, \quad \Gamma;\emptyset \vdash v : V, \quad \Gamma \sqcap u : V \sqcap v : U;\Delta \vdash_p P, \quad \Gamma;\Delta \vdash_p Q}{\Gamma;\Delta \vdash_p \text{if } (u = v) \text{ then } P \text{ else } Q}$$

(Check)

$$\frac{\begin{array}{ll}\Delta(n) = Nonce(p), \quad \Gamma;\emptyset \vdash u : U, \quad \Gamma;\emptyset \vdash n : V, & \\ \Gamma \sqcap u : V \sqcap n : U; (\Delta \sqcap u : Checked(p))[n : Checked(p)] \vdash_p P & \Gamma;\Delta \vdash_p Q\end{array}}{\Gamma;\Delta \vdash_p \text{check } (u = n) \text{ then } P \text{ else } Q}$$

involves two operations on the effect environment Δ: $\Delta \sqcap u : \tilde{E}$ is defined similarly to $\Gamma \sqcap u : U$, whereas $\Delta[n : \tilde{E}]$ changes the current association of n to $\tilde{E}$. The intuition underlying nonce verification is the following: consider the process $c(- : x, y).check\ (x = n)\ then\ P\ else\ Q$, run by the principal p, where n has the effect $Nonce(p)$. Then, the continuation P is type-checked in an environment where n's effect is consumed, and turned to $Checked(p)$, and x is deemed $Checked(p)$. Now, since $y : With(x, y)$, we can infer $y : Fresh^p$.

The remaining rules for processes and networks are reported in Table 6. The (Parallel) rule splits the Δ part of the environment between the parallel processes P and Q to avoid that the same nonce is checked by both P and Q; the disjoint union $\Delta_1 \uplus \Delta_2$ indicates the environment $\Delta_1 \sqcap \Delta_2$ in case $Nonces(\Delta_1) \cap Nonces(\Delta_2) = \emptyset$, and is undefined otherwise. The condition $Nonce(\Gamma;\Delta) = \emptyset$

Table 6 Other typing rules of processes and networks

(New)
$$\frac{\Gamma, a : T; \Delta' \vdash_p P, \quad T/\tilde{E} \textit{ generative for } p, \quad [\Delta' = \textit{if } \tilde{E} \neq \emptyset \textit{ then } (\Delta, a : \tilde{E}) \textit{ else } \Delta]}{\Gamma; \Delta \vdash_p (\nu a : T/\tilde{E})P}$$

(Parallel)
$$\frac{\Gamma; \Delta_1 \vdash_p P, \quad \Gamma; \Delta_2 \vdash_p Q}{\Gamma; \Delta_1 \uplus \Delta_2 \vdash_p P|Q}$$

(Recursion)
$$\frac{Nonces(\Gamma; \Delta) = \emptyset, \quad \Gamma, X : Proc; \Delta \vdash_p P}{\Gamma; \Delta \vdash_p \text{rec } X.P}$$

(Principal)
$$\frac{\Gamma; \Delta \vdash_p P}{\Gamma; \Delta \vdash p[\![P]\!]}$$

(Cache)
$$\frac{\Gamma; \Delta \vdash \diamond}{\Gamma; \Delta \vdash \overline{c}@q\langle \underline{p} : \tilde{m} \| \tilde{n} \rangle_i^\circ}$$

(Network Parallel)
$$\frac{\Gamma; \Delta_1 \vdash M, \quad \Gamma; \Delta_2 \vdash N}{\Gamma; \Delta_1 \uplus \Delta_2 \vdash M|N}$$

(Zero)
$$\frac{\Gamma; \Delta \vdash \diamond}{\Gamma; \Delta \vdash_p 0}$$

(Proc)
$$\frac{\Gamma; \Delta \vdash X : Proc}{\Gamma; \Delta \vdash_p X}$$

(Network Zero)
$$\frac{\Gamma; \Delta \vdash \diamond}{\Gamma; \Delta \vdash \mathbf{0}}$$

(Network New)
$$\frac{\Gamma, a : T; \Delta' \vdash M, \quad T/\tilde{E} \textit{generative for } M, \quad [\Delta' = \textit{if } \tilde{E} \neq \emptyset \textit{ then } (\Delta, a : \tilde{E}) \textit{ else } \Delta]}{\Gamma; \Delta \vdash (\nu a : T/\tilde{E})M}$$

in (Recursion) similarly ensures that a nonce is never checked more than once: it is necessary since the body of a recursive process can be instantiated multiple times, but it is type-checked only once [17]. In the (New) rules we restrict the possible types for new names to generative tagged types to control the way new names are introduced at run-time. The remaining rules are fairly standard.

4 Properties of the Type System

We start our analysis of the type system properties with subject reduction.

4.1 Subject Reduction

This is a standard result, but its formulation in our type system is more elaborate than usual, due to the structure of our types and effects and our LTS characterization of the operational semantics.

Substitution. As we already said, types are preserved during computation, while effects are not. In particular, the authenticity effects *From*, *Fresh* and

With associated with an input variable are not preserved by the substitution of that variable with the name received.

This is reflected in the following formulation of the substitution lemma, which as usual is crucial in the proof of subject reduction. Given a set of effects $\tilde{E}$, let $|\tilde{E}|$ denote the *effect-erasure* of E, that is the subset of E resulting from erasing all occurrences of the effects *From*, *Fresh* and *With* from $\tilde{E}$.

Lemma 1 (Substitution).

(i) *If* $\Gamma, x : T \sqcap \Gamma'; \Delta, x : \tilde{E} \sqcap \Delta' \vdash u : U/\tilde{F}$ *and* $\Gamma; \Delta \vdash n : T/|\tilde{E}|$, *then* $\Gamma \sqcap \Gamma'\{n/x\}; \Delta \sqcap \Delta'\{n/x\} \vdash u\{n/x\} : (U/|\tilde{F}|)\{n/x\}$;

(ii) *If* $\Gamma, x : T \sqcap \Gamma'; \Delta, x : \tilde{E} \sqcap \Delta' \vdash_p P$ *and* $\Gamma; \Delta \vdash n : T/|\tilde{E}|$, *then* $\Gamma \sqcap \Gamma'\{n/x\}; \Delta \sqcap \Delta'\{n/x\} \vdash_p P\{n/x\}$.

Based on this result, we may show that typing is preserved by each of the transitions in our LTS. Since some of the transitions may introduce fresh names, typing the derivative of a transition requires a new type environment that depends on the transition itself. We use the notation $(\Gamma; \Delta)$ *after* α to refer to such environments. Given a judgement $\Gamma; \Delta \vdash M$ and a transition $M \xrightarrow{\alpha} M'$, the environment $(\Gamma; \Delta)$ *after* α has a straightforward, but lengthy definition[3] (cf. Appendix A). To illustrate, assume $\Gamma; \Delta \vdash M$ and consider a transition $M \xrightarrow{c@p(-:\tilde{m},\tilde{n})} M'$, with $\Gamma; \Delta \vdash c : Chan\langle \tilde{T}; \tilde{U} \rangle$, and $|\tilde{m}| = |\tilde{T}|$, $|\tilde{n}| = |\tilde{U}|$. Then $(\Gamma; \Delta)$ *after* α is the environment $\Gamma \sqcap \tilde{m} : Public \sqcap \tilde{n} : Any; \Delta$, which refines the information on the types of the names $\tilde{m}$ and $\tilde{n}$ received.

Admissible transitions. A final technical subtlety is that the construction of the type environment $(\Gamma; \Delta)$ *after* α may fail (and the resulting environment be undefined). This may happen, for instance, after the input transition discussed above, as the meet operation involved in the construction of $(\Gamma; \Delta)$ *after* α fails if the names $\tilde{m}$ transmitted on c are already known to the environment at a *Secret* type (*Secret* $\sqcap$ *Public* is undefined). We rule out such transitions as *non-admissible*, in the following sense.

Definition 2 (Admissible Transition). *We say that a transition* $M \xrightarrow{\alpha} M'$ *is admissible for* $\Gamma; \Delta$, *written* $\Gamma; \Delta \vdash \alpha$, *if, whenever* $\alpha = c@q(\underline{p} : \tilde{m}, \tilde{n})$ *with* $\Gamma; \Delta \vdash c : Chan\langle \tilde{T}; \tilde{U} \rangle$ *and* $|\tilde{m}| = |\tilde{T}|$, $|\tilde{n}| = |\tilde{U}|$, *one has* $\tilde{m} \cap Secrets(\Gamma; \Delta) = \emptyset$ *and* $\underline{p} = -$.

If $M \xrightarrow{\alpha} M'$ is admissible for $(\Gamma; \Delta)$, it is easy to show that $(\Gamma; \Delta)$ *after* α is always defined, as the definition rules out the transitions that would pass secrets for the non-secrets positions of an input prefix.

Theorem 3 (Subject Reduction). *Assume* $\Gamma; \Delta \vdash M$, *and let* $M \xrightarrow{\alpha} M'$ *be admissible for* $\Gamma; \Delta$. *Then* $(\Gamma; \Delta)$ *after* $\alpha \vdash M'$.

[3] The notation is loose here, as indeed the definition needs to consider the reduction step $M \xrightarrow{\alpha} M'$ and not just the transition label α: hence, strictly speaking, we should rather say $(\Gamma; \Delta)$ *after* α *in* $M \xrightarrow{\alpha} M'$.

It is important to remark that the restriction to admissible transitions does not involve any loss of expressive power for the attacker. Notice to this regard that well-typed principals only have admissible transitions and, by Proposition 5 below, we know that they never leak their secrets. Hence, restricting to admissible transitions only amounts to assume that the attacker cannot impersonate any trusted principal, and does not know the initial secrets shared by the principals. This is a sound initial assumption for any network and, by Theorem 3, it is a property that is preserved by well-typed networks (there is no circularity here, as the proof of Proposition 5 does not rely on Theorem 3).

4.2 Secrecy

We first show that well-typed networks do not leak their secrets. Following [10], we first define what it means to leak an unrestricted secret.

Definition 4 (Revelation). *Let* $N \equiv N' \mid p[\![\overline{c}@q\langle \underline{p} : \tilde{m}\rangle^{\circ} \mid P]\!]$ *and take* $(\Gamma;\Delta)$ *and* s *such that* $\Gamma;\Delta \vdash N$ *and* $\Gamma;\Delta \vdash s : \mathit{Secret}(\tilde{r})$. *We say that* N *reveals* s *iff* $s = c$, *or* $s \in \tilde{m}$ *and either* $q \notin \tilde{r}$ *or* $\circ \neq \bullet$. *We say that* N *reveals a secret of* $(\Gamma;\Delta)$ *if* N *reveals* s *for some* $s \in \mathit{Secrets}(\Gamma;\Delta)$.

The definition readily extends to the general case when a secret may be restricted. Let $\Gamma;\Delta \vdash N$ with $N \equiv (\nu \tilde{a} : \tilde{\tau})N'$. N leaks a secret iff N' reveals a secret of $\Gamma, \tilde{a} : \tilde{\tau}_T; \Delta, \tilde{a} : \tilde{\tau}_E$. In other words, a network leaks a secret whenever it either outputs it in clear, or sends it to a principal outside the secrecy group, or uses it as a channel (the name of a channel is always leaked upon output).

Proposition 5 (Group Secrecy). *Assume* $\Gamma;\Delta \vdash M$. *Then* M *does not leak any secret.*

The proof of this proposition follows directly by an inspection of the typing rules. By Theorem 3, we then know that well-typed networks do not leak their reveal at any step of computation. Indeed, as we show next, the type system provides stronger secrecy guarantees, in that it prevents any (possibily implicit or indirect flow of secret information. As in [1] we formalize strong secrecy in terms of behavioral equivalence, which in turn we define based on the bisimilarity relation that results from our LTS semantics. Given a transition $M \xrightarrow{\alpha} M'$ we let $\hat{\alpha}$ denote α with the type annotations stripped away. Again, we restrict to admissible transitions, with respect to the secrecy assumptions provided by a typing environment.

Definition 6 (Bisimilarity). *A symmetric relation* $\mathscr{R}$ *between networks is a* $(\Gamma;\Delta)$*-bisimulation if whenever* $M \mathscr{R} N$ *and* $M \xrightarrow{\alpha} M'$ *with* $\Gamma;\Delta \vdash \alpha$, *there exists* N' *such that* $N \xrightarrow{\alpha'} N'$ *with* $\Gamma;\Delta \vdash \alpha'$, $\hat{\alpha} = \hat{\alpha}'$ *and* $M' \mathscr{R} N'$. $(\Gamma;\Delta)$*-bisimilarity, noted* $\sim_{(\Gamma;\Delta)}$, *is the largest* $(\Gamma;\Delta)$*-bisimulation.*

Theorem 7 (Strong Secrecy). *Let* $\Gamma;\Delta \vdash N$. *Then* $N \sim_{(\Gamma;\Delta)} N\sigma$ *for all injective substitutions* σ *of the names in* $\mathit{Secrets}(\Gamma;\Delta)$.

Notice the technical difference from the original characterization in [1]: in that case secrecy is characterized as the inability to tell networks apart based on the names of type *Any* they exchange. Our present formulation is directly based on secret names, instead.

4.3 Authenticity

As anticipated at the outset, we formalize authenticity by establishing a form of injective agreement between statements and expectations in the style of [13]. We start by introducing a new construct to express the authenticity expectations about an input variable. The syntax of processes is extended as follows, where $E \in \{From^p(\tilde{u}), Fresh^p(\tilde{u})\}$.

$$P, Q, R \ ::= \ \ldots \text{as in } \S 2 \ldots \mid \text{expect}\langle x : E\rangle.P$$

The expect$\langle\cdot\rangle$ prefix is not a binder, hence the variable x in expect$\langle x : E\rangle.P$ must be bound by an enclosing input prefix. The prefix form expect$\langle x : E\rangle.P$ is well-typed in a given type environment if the effect expected for x is consistent with the effects associated with x in the given environment (and the continuation P is well-typed). When x gets substituted by a name, as in expect$\langle m : E\rangle.P$, the prefix is disregarded (as names are never bound to authenticity effects).

Table 7 Typing rules for expect$\langle\cdot\rangle$

$$\frac{\Gamma; \Delta \vdash x : E, \quad \Gamma; \Delta \vdash_p P}{\Gamma; \Delta \vdash_p \text{expect}\langle x : E\rangle.P} \qquad \frac{\Gamma; \Delta \vdash_p P}{\Gamma; \Delta \vdash_p \text{expect}\langle m : E\rangle.P}$$

At run-time, expectations play the role of assertions that express the authenticity properties of a message exchange: specifically, the intention is to check that, in any run of a process, each expectation, say expect$\langle m : E\rangle$, can be associated with a previous output of m that validates the authenticity property on the exchange of m as expressed by the effect E. In particular, an expectation of an authentic message from p may not only be justified by a certified output from p (as one would certainly expect), but also by an anonymous output that includes in its payload a shared secret between the receiver and p, since secrets are not leaked in well-typed networks. Similarly, an expectation of a fresh, authentic message additionally requires the presence of a nonce in case the justifying output is anonymous, since we are interested in preventing replay attacks.

To formalize the justification mechanisms we just illustrated, we annotate run-time network configurations so as to connect each expect$\langle\cdot\rangle$ statement with the input prefix that binds the variable predicated by the statement. Since the run-time input transitions of a network are triggered by corresponding (Replay/Forward) transitions, we create the desired association by annotating the

expect prefix with the index i of the (i) transition induced by the input that binds the variable predicated by the prefix. Note that, in doing so, we actually link by the index i each received value predicated by an expect with a previous output containing that value, as desired.

We formalize all this by introducing an indexed version of the LTS from §2, which we derive by modifying the (Input), (Replay) and (Forward) rules in Table 1 as in Table 8, and introducing a new rule for the expect prefix.

Table 8 Indexed LTS

$$\text{(Input)}\quad \frac{\sigma = \{\tilde{m}/\tilde{x}\}}{p[\![c(\underline{q} : \tilde{x})^{\circ}.P]\!] \xrightarrow{c@p(\underline{q}:\tilde{m})^{\circ}_{i}} p[\![idx(P, i, \tilde{x})\sigma]\!]} \qquad \text{(Expect)}\quad \frac{}{p[\![\text{expect}^{(i)}\langle m : E\rangle.P]\!] \xrightarrow{(i)p\ expects\langle m:E\rangle} p[\![P]\!]}$$

$$\text{(Forward)}\quad \frac{N \equiv (\nu\tilde{a} : \tilde{\tau})(\hat{N}|\overline{c}@q\langle p : \tilde{m}\|\tilde{n}\rangle^{\circ}_{i}),\quad \hat{N} \xrightarrow{c@q(p:\tilde{m})^{\circ}_{i}} N'}{N \xrightarrow{(i)} (\nu\tilde{a} : \tilde{\tau})N'}$$

$$\text{(Replay)}\quad \frac{N \equiv (\nu\tilde{a} : \tilde{\tau})(\hat{N}|\overline{c}@q\langle - : \tilde{m}\|\tilde{n}\rangle^{\circ}_{i}),\quad \hat{N} \xrightarrow{c@q(-:\tilde{m})^{\circ}_{i}} N'}{N \xrightarrow{(i)} (\nu\tilde{a} : \tilde{\tau})(N'|\overline{c}@q\langle - : \tilde{m}\|\tilde{n}\rangle^{\circ}_{i})}$$

The input label in the premise of the (Replay) and (Forward) rules now conveys the index i of the induced replay/forward transition, while the (Input) rule states that, before applying the substitution σ, every expect$\langle\cdot\rangle$ in the continuation, whose variable will become instantiated, must be indexed with i. The goal is achieved via a function idx, defined by structural induction on processes (cf. Appendix A). The indexed occurrences of expect$\langle\cdot\rangle$ type-check just as the unindexed occurrences (cf. Table 7).

We formalize the intended correspondence between expectations and output messages by means of the following, somewhat elaborate definition.

Definition 8 (Justified Expectation). *Given a type environment $(\Gamma; \Delta)$, consider the sequence of reductions $N_0 \xrightarrow{\alpha_0} N_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{m-1}} N_m$. Let then $(\Gamma_k; \Delta_k)$ be the environment $(\Gamma; \Delta)$ after $\alpha_0, \ldots, \alpha_{k-1}$ for each $k \in [1..m]$, where α_k is admissible for $\Gamma_k; \Delta_k$, and let $\alpha_j = (i)p\ expects\langle m : E\rangle$. We say that α_j is justified by α_h $(h < j)$ if $\alpha_h = (i, \tilde{b} : \tilde{\tau})\overline{c}@p\langle\underline{q} : \tilde{m}\rangle^{\circ}_{i}$ and either of the following conditions holds:*

– *if* $E = From^p(\tilde{q})$*, then either* $\underline{q} \in \tilde{q}$ *or* $N_{h+1} \equiv (\nu\tilde{a} : \tilde{\tau})(N' \mid \overline{c}@p\langle \underline{q} : \tilde{m} \| \tilde{n} \rangle_i^\circ)$ *and there exists* $s \in \tilde{m}$ *such that* $\Gamma_h, \tilde{a} : \tilde{\tau}.T; \Delta_h, \tilde{a} : \tilde{\tau}.E \vdash s : Secret(p, \tilde{r})$ *with* $\tilde{r} \subseteq \tilde{q}$*;*
– *if* $E = Fresh^p(\tilde{q})$*, the conditions of the previous item hold and, when* $\underline{q} = -$*, we additionally require that* $\Gamma_h, \tilde{a} : \tilde{\tau}.T; \Delta_h, \tilde{a} : \tilde{\tau}.E \vdash n : Nonce(p)$ *for some* $n \in \tilde{m}$*.*

This notion of justification is clearly inspired by the correspondence assertions proposed in [19], and akin to similar definitions employed in companion type systems for authentication and authorization [14, 13]: an "expect" can in fact be seen as an *end* and its corresponding output as a *begin*. However, note that the previous definition allows us to characterize authenticity directly as a property of the data exchanged during a protocol rather than indirectly by extracting and interpreting the effects the protocol has on that data.

Theorem 9 (Authenticity). *Let* $\Gamma; \emptyset \vdash N$ *and* $N \equiv N_0 \xrightarrow{\alpha_0} N_1 \cdots \xrightarrow{\alpha_{m-1}} N_m$ *be a sequence of reductions with* α_k *admissible for* $(\Gamma_k; \Delta_k)$ *(*$k \in [1..m]$*). Then every expect label in the reduction sequence is justified by a previous output label in the sequence.*

5 Typing a Variant of 2KP

We show the type system at work on a simplified variant of the e-payment protocol 2KP [5] by IBM. This is a significant example, because the protocol relies on two different authentication schemes: a signature mechanism and the presentation of a shared secret. The latter kind of authentication is important, as in real settings it is unlikely that every principal possesses a signing key.

We can describe the protocol as follows. The customer *cust* sends the description *desc* of the order to the merchant *merc* along the channel *init*. The merchant checks that the description received is the expected one and creates a new transaction identifier *tid*. The merchant sends back the description, packaged with *tid*, to the customer via channel *invc*, and then sends a request to the acquirer *bank* along the channel *req*, providing *tid* and the price of the order. At this stage, the customer allows the payment to the merchant by a communication along *paym* with the acquirer, where it provides *tid*, price and credit card information *can*. Finally, the acquirer checks that the two requests (from the customer and the merchant) agree and that the credit card details provided by the customer are right. If so, the acquirer clears the transaction and sends a notification to the customer and the merchant. The protocol can be coded as the network $2KP = N_{cust} \mid N_{merc} \mid N_{bank}$, where each N_i is defined in Table 9. We use composed conditional guards as syntactic sugar for a series of nested conditionals. The $2KP$ network is well-typed under the following assumptions:

$$desc : Public, can : Secret(cust, bank), price : Secret(cust, merc, bank)$$

The type of the communication channels is immediately derived from the types of the exchanged data, while *cust*, *merc* and *bank* must be given type $Prin$. Below, we detail the most interesting aspects in the type derivation for the example.

Table 9 2KP Protocol Specification

$$
\begin{aligned}
N_{cust} \triangleq cust[\![\ & \overline{init}@merc\langle - : desc\rangle \mid invc(merc : x_{desc}, x_{tid}).\text{if } (desc = x_{desc}) \\
& \text{then } \overline{paym}@bank\langle - : x_{tid}, price, can\rangle^{\bullet} \mid conf(bank : x_{auth}).0\ ]\!] \\
N_{merc} \triangleq merc[\![\ & init(- : y_{desc}).\text{if } (y_{desc} = desc) \\
& \text{then } (\nu\ tid : Public)(\overline{invc}@cust\langle * : y_{desc}, tid\rangle \mid \\
& \overline{req}@bank\langle * : tid, price\rangle^{\bullet}) \mid resp(bank : y_{auth}).0\ ]\!] \\
N_{bank} \triangleq bank[\![\ & req(merc : z_{tid}, z_{price})^{\bullet}.paym(- : z'_{tid}, z'_{price}, z_{can})^{\bullet}. \\
& \text{if } (can = z_{can}\ \wedge z_{tid} = z'_{tid}\ \wedge z_{price} = z'_{price}) \\
& \text{then expect}\langle z_{tid} : Fresh^{bank}(merc)\rangle. \\
& \text{expect}\langle z'_{tid} : From^{bank}(cust)\rangle. \\
& (\overline{resp}@merc\langle * : z_{tid}\rangle \mid \overline{conf}@cust\langle * : z_{tid}\rangle)\ ]\!]
\end{aligned}
$$

First, since *merc* is able to certify its messages, *bank* can easily derive that the expedition of z_{tid} is effectively from *merc*, since that variable is closed by a signed input. Once again, notice that this communication mode guarantees the freshness of the message. Secondly, even though z_{can} is closed by a non-certified input, the receiver can use that variable with type $Secret(cust, bank)$ and not with type *Any* to type-check the continuation, since z_{can} is checked against *can* and the typing rule for the conditional branch refines the types for the considered values upon a successful match; Finally, although *cust* has not a mean to certify its messages, the expedition of z'_{tid} must come from *cust*, since $z'_{tid} : With(z_{can})$ and $z_{can} : Secret(cust, bank)$ by the previous point. Note that, even if $z'_{tid} = z_{tid}$ and z_{tid} is fresh, the type-checker cannot assert the freshness of z'_{tid}. Even if this may seem limitative in this particular case, given the nature of the tid, this is the sound choice to make, since in general it is unsound to infer the freshness of a piece of data from its equality with a fresh value.

We remark that the type-checking is completely syntax-directed and compositional, yielding a rather effective tool for protocol verification. In spite of this simplicity, the type system turns out to be quite expressive and flexible, using the correlation rules to infer authenticity information and the conditional branches to refine the types of the values.

6 Conclusions

The analysis of distributed systems built upon secure channel abstractions has been subject of active research in the recent literature, based on various formalisms: model checking [3], CSP-style traces specifications [12], Strand spaces [16], inductive verification [4] and process calculi [2, 11].

The present paper continues on the line of work initiated in [7, 6], by introducing a type system to provide static security guarantees for the high-level abstractions for distributed messaging proposed in those papers.

The type system enforces two main security guarantees – strong secrecy and strong authenticity in the presence of a Dolev-Yao intruder – which are comparable to those provided by companion type systems for security. The novelties of our approach are mainly technical, but they also bear conceptual significance. In particular, the ability to characterize authenticity as a property of data itself, which is distinctive of our type system, appears to constitute an important step towards the integration of security types within typing systems for (semi) structured datatypes available in modern programming languages. This kind of integration within languages accommodating more structured interaction primitives such as those available in session description languages and/or choreography languages represent one of the lines of work we plan for our future research.

References

1. Abadi, M.: Secrecy by typing in security protocols. Journal of the ACM 46(5), 749–786 (1999)
2. Adão, P., Fournet, C.: Cryptographically sound implementations for communicating processes. In: ICALP (2). pp. 83–94 (2006)
3. Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Tobarra, M.L.: Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In: FMSE. pp. 1–10 (2008)
4. Bella, G., Longo, C., Paulson, L.C.: Verifying second-level security protocols. In: TPHOLs. pp. 352–366 (2003)
5. Bellare, M., Garay, J.A., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Herreweghen, E.V., Waidner, M.: Design, implementation, and deployment of the iKP secure electronic payment system. IEEE Journal on Selected Areas in Communications 18, 611–627 (2000)
6. Bugliesi, M., Focardi, R.: Channel abstractions for network security. Mathematical Structures in Computer Science xx, xxx–xxx (2010)
7. Bugliesi, M., Focardi, R.: Language based secure communication. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008. pp. 3–16. IEEE Computer Society (2008)
8. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. ACM Trans. Comput. Syst. 8(1), 18–36 (1990)
9. Calzavara, S.: Security Types for Distributed Applications. Master's thesis, Università Ca' Foscari di Venezia (2009)
10. Cardelli, L., Ghelli, G., Gordon, A.D.: Secrecy and group creation. Inf. Comput. 196(2), 127–155 (2005)
11. Corin, R., Deniélou, P.M., Fournet, C., Bhargavan, K., Leifer, J.J.: Secure implementations for typed session abstractions. In: 20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy. pp. 170–186. IEEE Computer Society (2007)
12. Dilloway, C., Lowe, G.: Specifying secure transport channels. In: CSF. pp. 210–223 (2008)

13. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. ACM Trans. Program. Lang. Syst. 29(5) (2007)
14. Gordon, A.D., Jeffrey, A.: Types and effects for asymmetric cryptographic protocols. Journal of Computer Security 12(3-4), 435–483 (2004)
15. Hennessy, M., Rathke, J.: Typed behavioural equivalences for processes in the presence of subtyping. Mathematical Structures in Computer Science 14(5), 651–684 (2004)
16. Kamil, A., Lowe, G.: Specifying and modelling secure channels in strand spaces. In: Workshop on Formal Aspects of Security and Trust (FAST) (2009)
17. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. ACM Trans. Program. Lang. Syst. 21(5), 914–947 (1999)
18. Merro, M., Sangiorgi, D.: On asynchrony in name-passing calculi. Mathematical Structures in Computer Science 14(5), 715–767 (2004)
19. Woo, T.Y.C., Lam, S.S.: A semantic model for authentication protocols. In: IEEE Symposium on Security and Privacy. p. 178 (1993)

A Appendix

The function *after*. The function $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M')$ is partially defined as follows:

- if $\alpha = c(- : \tilde{m}, \tilde{n})$ with $\Gamma;\Delta \vdash c : Chan\langle\tilde{T};\tilde{U}\rangle$, $|\tilde{m}| = |\tilde{T}|$ and $|\tilde{n}| = |\tilde{U}|$, then $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M') = (\Gamma \sqcap \tilde{m} : Public \sqcap \tilde{n} : Any; \Delta)$;
- if $\alpha = c(p : \tilde{m}, \tilde{n})$ with $\Gamma;\Delta \vdash c : Chan\langle\tilde{T};\tilde{U}\rangle$, $|\tilde{m}| = |\tilde{T}|$ and $|\tilde{n}| = |\tilde{U}|$, then $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M') = (\Gamma \sqcap \tilde{m} : \tilde{T} \sqcap \tilde{n} : \tilde{U}; \Delta)$;
- if $\alpha = (i, \tilde{b} : \tilde{\tau})\overline{c}@p\langle\underline{q} : \tilde{m}\rangle_i^\circ$ with $b_j : \tau_j = T_j/\tilde{E}_j$ for each j, then $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M') = (\Gamma, \tilde{b} : \tilde{T}; \Delta')$ with $\Delta' = \Delta \cup \{b_j : Nonce(q) \mid \tilde{E}_j = Nonce(q)\}$;
- if $\alpha = (i)$, then $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M') = (\Gamma \sqcap \tilde{m} : Public \sqcap \tilde{n} : Any; \Delta)$ if $i \in fn(M')$, $(\Gamma;\Delta)$ *after* α *in* $(M \xrightarrow{\alpha} M') = (\Gamma \sqcap \tilde{m} : \tilde{T} \sqcap \tilde{n} : \tilde{U}; \Delta)$ otherwise.

The definition exploits the fact that a message cached at i is consumed by a transition (i) if and only if it is signed by a principal.

The function *idx*. The function $idx(P, i, \tilde{y})$ is defined by induction on the structure of the process P as follows:

- $idx(\text{expect}\langle x : E\rangle.P, i, \tilde{y}) = \text{expect}^{(i)}\langle x : E\rangle.idx(P, i, \tilde{y})$, if $x \in \tilde{y}$;
- $idx(\text{expect}\langle x : E\rangle.P, i, \tilde{y}) = \text{expect}\langle x : E\rangle.idx(P, i, \tilde{y})$, if $x \notin \tilde{y}$;
- $idx(\overline{u}@v\langle\underline{*} : \tilde{u}\rangle^\circ, i, \tilde{y}) = \overline{u}@v\langle\underline{*} : \tilde{u}\rangle^\circ$;
- $idx(a(\underline{u} : \tilde{x})^\circ.P, i, \tilde{y}) = a(\underline{u} : \tilde{x})^\circ.idx(P, i, \tilde{y})$;
- $idx(0, i, \tilde{y}) = 0$;
- $idx(P|Q, i, \tilde{y}) = idx(P, i, \tilde{y})|idx(Q, i, \tilde{y})$;
- $idx(\text{test}(u = v) \text{ then } P \text{ else } Q, i, \tilde{y}) = \text{test}(u = v) \text{ then } idx(P, i, \tilde{y}) \text{ else } idx(Q, i, \tilde{y})$;
- $idx(\text{rec } X.P, i, \tilde{y}) = \text{rec } X.idx(P, i, \tilde{y})$;
- $idx(X, i, \tilde{y}) = X$;
- $idx((\nu a : \tau)P, i, \tilde{y}) = (\nu a : \tau)idx(P, i, \tilde{y})$.