Type-Based Automated Verification of Authenticity in Asymmetric Cryptographic Protocols

Morten Dahl[2], Naoki Kobayashi[1], Yunde Sun[1], and Hans Hüttel[2]

[1] Tohoku University
[2] Aalborg University

Abstract. Gordon and Jeffrey developed a type system for verification of asymmetric and symmetric cryptographic protocols. We propose a modified version of Gordon and Jeffrey's type system and develop a type inference algorithm for it, so that protocols can be verified automatically as they are, without any type annotations or explicit type casts. We have implemented a protocol verifier SPICA2 based on the algorithm, and confirmed its effectiveness.

1 Introduction

Security protocols play a crucial role in today's Internet technologies including electronic commerce and voting. Formal verification of security protocols is thus an important, active research topic, and a variety of approaches to (semi-)automated verification have been proposed [8, 5, 16]. Among others, type-based approaches [1, 15, 16] have advantages that protocols can be verified in a modular manner, and that it is relatively easy to extend them to verify protocols at the source code level [4]. They have however a disadvantage that users have to provide complex type annotations, which require expertise in both security protocols and type theories. Kikuchi and Kobayashi [19] developed a type inference algorithm but it works only for symmetric cryptographic protocols.

To overcome the limitation of the type-based approaches and enable fully automated protocol verification, we integrate and extend the two lines of work – Gordon and Jeffrey's work [16] for verifying protocols using both symmetric and asymmetric cryptographic protocols, and Kikuchi and Kobayashi's work. The outcome is an algorithm for automated verification of authenticity in symmetric and asymmetric cryptographic protocols. The key technical novelty lies in the symmetric notion of *obligations* and *capabilities* attached to name types, which allows us to reason about causalities between actions of protocol participants in a general and uniform manner in the type system. It not only enables automated type inference, but also brings a more expressive power, enabling, e.g., verification of multi-party cryptographic protocols. We have developed a type inference algorithm for the new type system, and implemented a protocol verification tool SPICA2 based on the algorithm. According to experiments, SPICA2 is very fast; it could successfully verify a number of protocols in less than a second.

The rest of this paper is structured as follows. Section 2 introduces spi-calculus [2] extended with correspondence assertions as a protocol description language. Sections 3 and 4 present our type system and sketches a type inference algorithm. Section 5 reports implementation and experiments. Sections 6 and 7 discuss extensions and related work respectively. Section 8 concludes the paper.

2 Processes

This section defines the syntax and operational semantics of the spi-calculus extended with correspondence assertions, which we call spi$_{CA}$. The calculus is essentially the same as that of Gordon and Jeffrey [16], except (i) there are no type annotations or casts (as they can be automatically inferred by our type inference algorithm), and (ii) there are no primitives for witness and trust; supporting them is left for future work.

We assume that there is a countable set of *names*, ranged over by $m, n, k, x, y, z, \ldots$. By convention, we often use $k, m, n, \ldots$ for free names and $x, y, z, \ldots$ for bound names.

The set of messages, ranged over by M, is given by:

$$M ::= x \mid (M_1, M_2) \mid \{M_1\}_{M_2} \mid \{\!|M_1|\!\}_{M_2}$$

(M_1, M_2) is a pair consisting of M_1 and M_2. The message $\{M_1\}_{M_2}$ ($\{\!|M_1|\!\}_{M_2}$, resp.) represents the ciphertext obtained by encrypting M_1 with the symmetric (asymmetric, resp.) key M_2. For the asymmetric encryption, we do not distinguish between encryption and signing; $\{\!|M_1|\!\}_{M_2}$ denotes an encryption if M_2 is a public key, while it denotes signing if M_2 is a private key.

The set of processes, ranged over by P, is given by:

$$\begin{aligned}P ::= & \; \mathbf{0} \mid M_1!M_2 \mid M?x.P \mid (P_1 \mid P_2) \mid *P \mid (\nu x)P \mid (\nu_{sym} x)P \mid (\nu_{asym} x, y)P \\ & \mid \mathbf{check}\ M_1\ \mathbf{is}\ M_2.P \mid \mathbf{split}\ M\ \mathbf{is}\ (x, y).P \mid \mathbf{match}\ M_1\ \mathbf{is}\ (M_2, y).P \\ & \mid \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P \mid \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P \\ & \mid \mathbf{begin}\, M.P \mid \mathbf{end}\, M\end{aligned}$$

The names denoted by x, y are *bound* in P. We write $[M_1/x_1, \ldots, M_n/x_n]P$ for the process obtained by replacing every free occurrence of $x_1, \ldots, x_n$ in P with $M_1, \ldots,$ M_n. We write $\mathbf{FN}(P)$ for the set of free (i.e. non-bounded) names in P.

Process $\mathbf{0}$ does nothing, $M_1!M_2$ sends M_2 over the channel M_1, and $M_1?x.P$ waits to receive a message on channel M_1, and then binds x to it and behaves like P. $P_1 \mid P_2$ executes P_1 and P_2 in parallel, and $*P$ executes infinitely many copies of P in parallel.

We have three kinds of name generation primitives: (νx) for ordinary names, $(\nu_{sym} x)$ for symmetric keys, and $(\nu_{asym} x_1, x_2,)$ for asymmetric keys. $(\nu_{asym} x_1, x_2, P)$ creates a fresh key pair (k_1, k_2) (where k_1 and k_2 are encryption and decryption keys respectively), and behaves like $[k_1/x_1, k_2/x_2]P$. The process $\mathbf{check}\ M_1\ \mathbf{is}\ M_2.P$ behaves like P if M_1 and M_2 are the same name, and otherwise behaves like $\mathbf{0}$. The process $\mathbf{split}\ M\ \mathbf{is}\ (x, y).P$ behaves like $[M_1/x, M_2/y]P$ if M is a pair (M_1, M_2); otherwise it behaves like $\mathbf{0}$. $\mathbf{match}\ M_1\ \mathbf{is}\ (M_2, y).P$ behaves like $[M_3/y]P$ if M_1 is a pair of the form (M_2, M_3); otherwise it behaves like $\mathbf{0}$. Process $\mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P$ ($\mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P$, resp.) decrypts ciphertext M_1 with symmetric (asymmetric, resp.) key M_2, binds x to the result and behaves like P; if M_1 is not an encryption, or an encryption with a key not matching M_2, then it behaves like $\mathbf{0}$. The process $\mathbf{begin}\, M.P$ raise an event $\mathbf{begin}\, M$ and behaves like P, while $\mathbf{end}\, M$ just raises an event $\mathbf{end}\, M$; they are used to express expected authenticity properties.

Example 1. We use the three protocols in Figure 1, taken from [16], as running examples. POSH and SOSH protocols aim to pass a new message `msg` from B to A, so

POSH:	**SOPH**	**SOSH**
`A->B: n`	`A->B:` $\{\vert(\mathtt{msg},\mathtt{n})\vert\}_{pk_B}$	`A->B:` $\{\vert\mathtt{n}\vert\}_{pk_B}$
`B begins msg`	`B begins msg`	`B begins msg`
`B->A:` $\{\vert(\mathtt{msg},\mathtt{n})\vert\}_{sk_B}$	`B->A: n`	`B->A:` $\{\vert\mathtt{msg},\mathtt{n}\vert\}_{pk_A}$
`A ends msg`	`A ends msg`	`A ends msg`

Fig. 1. Informal Description of Three Protocols

$(\nu_{asym} sk_B, pk_B)(net!pk_B \mid$	(* create asymmetric keys for B and make pk_B public *)
$(\nu non)(net!non \mid$	(* A creates a nonce and sends it *)
$net?ctext.\mathbf{decrypt}\ ctext\ \mathbf{is}\ \{\!\vert x\vert\!\}_{pk_B^{-1}}.$	(* receive a cypertext and decrypt it*)
$\mathbf{split}\ x\ \mathbf{is}\ (m, non').\mathbf{check}\ non\ \mathbf{is}\ non'.$	(* decompose pair x and check nonce *)
$\mathbf{end}\ m) \mid$	(* believe that m came from B *)
$net?n.$	(* B receives a nonce *)
$(\nu msg)\mathbf{begin}\ msg.$	(* create a message and declare that it is going to be sent*)
$net!\{\!\vert(msg, n)\vert\!\}_{sk_B})$	(* encrypt and send (msg, n) *)

Fig. 2. Public-Out-Secret-Home (POSH) protocol in spi$_{CA}$

$(\nu_{asym} pk_B, sk_B)$	(* create asymmetric keys for B *)
$(net!pk_B$	(* make pk_B public *)
$\mid$	(* Behavior of A *)
$(\nu non)(\nu msg)$	(* create a nonce and a message *)
$(net!\{\!\vert(msg, non)\vert\!\}_{pk_B} \mid$	(* encrypt and send (msg, non) *)
$net?non'.$	(* receive a nonce *)
$\mathbf{check}\ non\ \mathbf{is}\ non'.$	(* check nonce *)
$\mathbf{end}\ msg)$	(* end assertion *)
$\mid$	(* Behavior of B *)
$net?ctext.$	(* receive a cypertext *)
$\mathbf{decrypt}\ ctext\ \mathbf{is}\ \{\!\vert x\vert\!\}_{sk_B^{-1}}.$	(* decrypt the cypertext *)
$\mathbf{split}\ x\ \mathbf{is}\ (m, non'').$	(* decompose pair x *)
$\mathbf{begin}\ m.$	(* begin assertion *)
$net!non''$	(* send the nonce *)
$)$	

Fig. 3. Secret-Out-Public-Home (SOPH) protocol in spi$_{CA}$

that A can confirm that `msg` indeed comes from B, while SOPH protocol aims to pass `msg` from A to B, so that A can confirm that `msg` has been received by B. The second and fourth lines of each protocol expresses the required authenticity by using Woo and Lam's correspondence assertions [21]. "`B begins msg`" on the second line of POSH means "B is going to send `msg`", and "`A ends msg`" on the fourth line means "A believes that B has sent `msg`". The required authenticity is then expressed as a correspondence between begin- and end-events: whenever an end-event ("`A ends msg`" in this example) occurs, the corresponding begin-event ("`B begins msg`") must have occurred.[3] In the three protocols, the correspondence between begin- and end-events is guaranteed in different ways. In POSH, the correspondence is guaranteed by the signing of the second message with B's secret key, so that A can verify that B has created the pair (msg, n). In SOPH, it is guaranteed by encrypting the first message with B's public key, so that the nonce n, used as an acknowledgment, cannot be forged by an attacker. SOSH is similar to POSH, but keeps `n` secret by using A and B's public keys.

Figure 2 gives a formal description of POSH protocol, represented as a process in spi$_{CA}$. The first line is an initial set-up for the protocol. An asymmetric key pair for B is created and the decryption key *pk*$_B$ is sent on a public channel *net*, on which an attacker can send and receive messages. The next four lines describe the behavior of A. On the second line, a nonce *non* is created and sent along *net*. On the third line, a ciphertext *ctext* is received and decrypted (or verified) with B's public key. On the fourth line, the pair is decomposed and it is checked that the second component coincides with the nonce sent before. On the fifth line, an end-event is raised, meaning that A believes that *msg* came from B. The last three lines describe the behavior of B. On the sixth line, a nonce n is received from *net*. On the seventh line, a new message *msg* is created and a begin-event is raised, meaning that B is going to send *msg*. On the last line, the pair (msg, n) is encrypted (or signed) with B's secret key and sent on *net*.

Figure 3 gives a formal description of SOPH protocol in spi$_{CA}$. □

Following Gordon and Jeffrey, we call a process *safe* if it satisfies correspondence assertions (i.e. for each end-event, a corresponding begin-event has occurred before), and *robustly safe* if a process is safe in the presence of arbitrary attackers (representable in spi$_{CA}$). Proving robust safety automatically is the goal of protocol verification in the present paper. To formalize the robust safety, we use the operational semantics shown in Figure 4. A runtime state is a quadruple $\langle \Psi, E, N, \mathcal{K} \rangle$, where Ψ is a multiset of processes, and E is the multiset of messages on which begin-events have occurred but the matching end-events have not. N is the set of names (including keys) created so far, and $\mathcal{K}$ is the set of key pairs. The special runtime state $\mathbf{Error}$ denotes that correspondence assertions have been violated. Note that a reduction gets stuck when a process does not match a rule. For example, $\mathbf{split}\ M\ \mathbf{is}\ (x, y).P$ is reducible only if M is of the form (M_1, M_2). Using the operational semantics, the robust safety is defined as follows.

[3] There are two types of correspondence assertions in the literature: non-injective (or one-to-many) and injective (or one-to-one) correspondence. Throughout the paper we consider the latter.

$$\langle \Psi \uplus \{n?y.P, n!M\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[M/y]P\}, E, N, \mathcal{K}\rangle \qquad \text{(R-COM)}$$

$$\langle \Psi \uplus \{P \mid Q\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{P, Q\}, E, N, \mathcal{K}\rangle \qquad \text{(R-PAR)}$$

$$\langle \Psi \uplus \{*P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{*P, P\}, E, N, \mathcal{K}\rangle \qquad \text{(R-REP)}$$

$$\langle \Psi \uplus \{(\nu x)P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[n/x]P\}, E, N \cup \{n\}, \mathcal{K}\rangle \ (n \notin N) \qquad \text{(R-NEW)}$$

$$\langle \Psi \uplus \{(\nu_{sym} x)P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[k/x]P\}, E, N \cup \{k\}, \mathcal{K}\rangle \ (k \notin N) \quad \text{(R-NEWSK)}$$

$$\begin{array}{l} \langle \Psi \uplus \{(\nu_{asym} x, y)P\}, E, N, \mathcal{K}\rangle \\ \longrightarrow \langle \Psi \uplus \{[k_1/x, k_2/y]P\}, E, N \cup \{k_1, k_2\}, \mathcal{K} \cup \{(k_1, k_2)\}\rangle \ (k_1, k_2 \notin N) \end{array} \qquad \text{(R-NEWAK)}$$

$$\langle \Psi \uplus \{\mathbf{check}\ n\ \mathbf{is}\ n.P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{P\}, E, N, \mathcal{K}\rangle \qquad \text{(R-CHK)}$$

$$\langle \Psi \uplus \{\mathbf{split}\ (M, N)\ \mathbf{is}\ (x, y).P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[M/x, N/y]P\}, E, N, \mathcal{K}\rangle \qquad \text{(R-SPLT)}$$

$$\langle \Psi \uplus \{\mathbf{match}\ (M, N)\ \mathbf{is}\ (M, z).P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[N/z]P\}, E, N, \mathcal{K}\rangle \qquad \text{(R-MTCH)}$$

$$\langle \Psi \uplus \{\mathbf{decrypt}\ \{M\}_k\ \mathbf{is}\ \{x\}_k.P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{[M/x]P\}, E, N, \mathcal{K}\rangle \quad \text{(R-DECS)}$$

$$\begin{array}{l} \langle \Psi \uplus \{\mathbf{decrypt}\ \{\!|M|\!\}_{k_1}\ \mathbf{is}\ \{\!|x|\!\}_{k_2{}^{-1}}.P\}, E, N, \mathcal{K}\rangle \\ \longrightarrow \langle \Psi \uplus \{[M/x]P\}, E, N, \mathcal{K}\rangle \ (\text{if } (k_1, k_2) \in \mathcal{K}) \end{array} \qquad \text{(R-DECA)}$$

$$\langle \Psi \uplus \{\mathbf{begin}\, M.P\}, E, N, \mathcal{K}\rangle \longrightarrow \langle \Psi \uplus \{P\}, E \uplus \{M\}, N, \mathcal{K}\rangle \qquad \text{(R-BGN)}$$

$$\langle \Psi \uplus \{\mathbf{end}\, M\}, E \uplus \{M\}, N, \mathcal{K}\rangle \longrightarrow \langle \Psi, E, N, \mathcal{K}\rangle \qquad \text{(R-END)}$$

$$\langle \Psi \uplus \{\mathbf{end}\, M\}, E, N, \mathcal{K}\rangle \longrightarrow \mathbf{Error} \quad (\text{if } M \notin E) \qquad \text{(R-ERR)}$$

Fig. 4. Operational Semantics

Definition 21 (safety, robust safety) *A process P is* safe *if $\langle \{P\}, \emptyset, \mathbf{FN}(P), \emptyset\rangle \not\longrightarrow^* \mathbf{Error}$. A process P is* robustly safe *if $P|O$ is safe for every spi$_{CA}$ process O that contains no begin/end/check operations.*[4]

3 Type System

This section presents a type system such that well-typed processes are robustly safe. This allows us to reduce protocol verification to type inference.

3.1 Basic Ideas

Following the previous work [15, 16, 19], we use the notion of *capabilities* (called effects in [15, 16]) in order to statically guarantee that end-events can be raised only after the corresponding begin-events. A capability φ is a multiset of *atomic capabilities* of the form $\mathbf{end}(M)$, which expresses a permission to raise "end M" event. The robust safety of processes is guaranteed by enforcing the following conditions on capabilities: (i) to raise an "end M" event, a process must possess and consume an atomic $\mathbf{end}(M)$ capability; and (ii) an atomic $\mathbf{end}(M)$ capability is generated only by raising a "begin M" event. Those conditions can be statically enforced by using a type judgment of the

[4] Having no check operations is not a limitation, as an attacker process can check the equality of n_1 and n_2 by $\mathbf{match}\ (n_1, n_1)\ \mathbf{is}\ (n_2, x).P$.

form: $\Gamma; \varphi \vdash P$, which means that P can be safely executed under the type environment Γ and the capabilities described by φ. For example, $x : T; \{\mathbf{end}(x)\} \vdash \mathbf{end}\, x$ is a valid judgment, but $x : T; \emptyset \vdash \mathbf{end}\, x$ is not. The two conditions above can be locally enforced by the following typing rules for begin and end events:

$$\frac{\Gamma; \varphi + \{\mathbf{end}(M)\} \vdash P}{\Gamma; \varphi \vdash \mathbf{begin}\, M.P} \qquad \frac{}{\Gamma; \varphi + \{\mathbf{end}(M)\} \vdash \mathbf{end}\, M}$$

The left rule ensures that the new capability $\mathbf{end}(M)$ is available after the begin-event, and the right rule for end ensures that the capability $\mathbf{end}(M)$ must be present.

The main difficulty lies in how to pass capabilities between processes. For example, recall the POSH protocol in Figure 2, where begin- and end-events are raised by different protocol participants. The safety of this protocol can be understood as follows: B obtains the capability $\mathbf{end}(\mathit{msg})$ by raising the begin event, and then passes the capability to A by attaching it to the nonce n. A then extracts the capability and safely executes the end event. As n is signed with B's private key, there is no way for an attacker to forge the capability. For another example, consider the SOPH protocol in the middle of Figure 1. In this case, the nonce n is sent in clear text, so that B cannot pass the capability to A through the second message. Instead, the safety of the SOPH protocol is understood as follows: A attaches to n (in the first message) an *obligation* to raise the begin-event. B then discharges the obligation by raising the begin-event, and notifies of it by sending back n. Here, note that an attacker cannot forge n, as it is encrypted by B's public key in the first message.

To capture the above reasoning by using types, we introduce types of the form $\mathbf{N}(\varphi_1, \varphi_2)$, which describes names carrying an obligation φ_1 and a capability φ_2. In the examples above, n is given the type $\mathbf{N}(\emptyset, \{\mathbf{end}(\mathit{msg})\})$ in the second message of POSH protocol, and the type $\mathbf{N}(\{\mathbf{end}(\mathit{msg})\}, \emptyset)$ in the first message of SOPH protocol.

The above types $\mathbf{N}(\emptyset, \{\mathbf{end}(\mathit{msg})\})$ and $\mathbf{N}(\{\mathbf{end}(\mathit{msg})\}, \emptyset)$ respectively correspond to *response* and *challenge types* in Gordon and Jeffrey's type system [16]. Thanks to the uniform treatment of name types, type inference for our type system reduces to a problem of solving constraints on capabilities and obligations, which can further be reduced to linear programming problems by using the technique of [19]. The uniform treatment also allows us to express a wider range of protocols (such as multi-party cryptographic protocols). Note that neither obligations nor asymmetric cryptography are supported by the previous type system for automated verification [19]; handling them requires non-trivial extensions of the type system and the inference algorithm.

3.2 Types

Definition 31 *The syntax of types, ranged over by τ, is given by:*

$$\begin{array}{lr}
\tau ::= \mathbf{N}_\ell(\varphi_1, \varphi_2) \mid \mathbf{SKey}(\tau) \mid \mathbf{DKey}(\tau) \mid \mathbf{EKey}(\tau) \mid \tau_1 \times \tau_2 & \\
\varphi ::= \{A_1 \mapsto r_1, \ldots, A_m \mapsto r_m\} & \textit{capabilities} \\
A ::= \mathbf{end}(M) \mid \mathbf{chk}_\ell(M, \varphi) & \textit{atomic cap.} \\
\iota ::= x \mid 0 \mid 1 \mid 2 \mid \cdots & \textit{extended names} \\
\ell ::= \mathbf{Pub} \mid \mathbf{Pr} & \textit{name qualifiers}
\end{array}$$

Here, r_i ranges over non-negative rational numbers.

The type $\mathbf{N}_\ell(\varphi_1, \varphi_2)$ is assigned to names carrying obligations φ_1 and capabilities φ_2. Here, obligations and capabilities are mappings from atomic capabilities to rational numbers. For example, $\mathbf{N}_\ell(\{\mathbf{end}(a) \mapsto 1.0\}, \{\mathbf{end}(b) \mapsto 2.0\})$ describes a name that carries the obligation to raise $\mathbf{begin}\, a$ once, and the capability to raise $\mathbf{end}\, b$ twice. Fractional values are possible: $\mathbf{N}_\ell(\emptyset, \{\mathbf{end}(b) \mapsto 0.5\})$ means that the name carries a half of the capability to raise $\mathbf{end}\, b$, so that if combined with another half of the capability, it is allowed to raise $\mathbf{end}\, b$. The introduction of fractions slightly increases the expressive power of the type system, but the main motivation for it is rather to enable efficient type inference as in [19]. When the ranges of obligations and capabilities are integers, we often use multiset notations; for example, we write $\{\mathbf{end}(a), \mathbf{end}(a), \mathbf{end}(b)\}$ for $\{\mathbf{end}(a) \mapsto 2, \mathbf{end}(b) \mapsto 1\}$. The atomic capability $\mathbf{chk}_\ell(M, \varphi)$ expresses the capability to check equality on M by $\mathbf{check}\ M\ \mathbf{is}\ M'.P$: since nonce checking releases capabilities this atomic effect is used to ensure that each nonce can only be checked once. The component φ expresses the capability that can be extracted by the check operation (see the typing rule for check operations given later).

Qualifier ℓ attached to name types are essentially the same as the **Public**/**Private** qualifiers in Gordon and Jeffrey's type system and express whether a name can be made public or not. We often write $\mathbf{Un}$ for $\mathbf{N}_{\mathbf{Pub}}(\emptyset, \emptyset)$.

The type $\mathbf{SKey}(\tau)$ describes symmetric keys used for decrypting and encrypting values of type τ. The type $\mathbf{EKey}(\tau)$ ($\mathbf{DKey}(\tau)$, resp.) describes asymmetric keys used for encrypting (decrypting, resp.) values of type τ. The type $\tau_1 \times \tau_2$ describes pairs of values of types τ_1 and τ_2. As in [19], we express the dependency of types on names by using indices. For example, the type $\mathbf{Un} \times \mathbf{N}_\ell(\emptyset, \{\mathbf{end}(0)\})$ denotes a pair (M_1, M_2) where M_1 has type $\mathbf{Un}$ and M_2 has type $\mathbf{N}_\ell(\emptyset, \{\mathbf{end}(M_1)\})$. The type $\mathbf{Un}\times(\mathbf{Un}\times\mathbf{N}_{\mathbf{Pub}}(\emptyset, \{\mathbf{end}(0, 1) \mapsto r\})$ describes triples of the form $(M_1, (M_2, M_3))$, where M_1 and M_2 have type $\mathbf{Un}$, and M_3 has type $\mathbf{N}_{\mathbf{Pub}}(\emptyset, \{\mathbf{end}(M_2, M_1) \mapsto r\})$. In general, an index i is a natural number referring to the i-th closest first component of pairs. In the syntax of atomic capabilities $\mathbf{end}(M)$, M is an extended message that may contain indices. We use the same metavariable M for the sake of simplicity.

Predicates on types Following Gordon and Jeffrey, we introduce two predicates $\mathbf{Pub}$ and $\mathbf{Taint}$ on types, inductively defined by the rules in Figure 5. $\mathbf{Pub}(\tau)$ means that a value of type τ can safely be made public by e.g. sending it through a public channel. $\mathbf{Taint}(\tau)$ means that a value of type τ may have come from an untrusted principal and hence cannot be trusted. It may for instance have been received through a public channel or have been extracted from a ciphertext encrypted with a public key.

The first rule says that for $\mathbf{N}_\ell(\varphi_1, \varphi_2)$ to be public, the obligation φ_1 must be empty, as there is no guarantee that an attacker fulfills the obligation. Contrary, for $\mathbf{N}_\ell(\varphi_1, \varphi_2)$ to be tainted, the capability φ_2 must be empty if $\ell = \mathbf{Pub}$, as the name may come from an attacker and the capability cannot be trusted.[5]

$\mathbf{Pub}$ and $\mathbf{Taint}$ are a sort of dual, flipped by the type constructor $\mathbf{EKey}$. In terms of subtyping, $\mathbf{Pub}(\tau)$ and $\mathbf{Taint}(\tau)$ may be understood as $\tau \leq \mathbf{Un}$ and $\mathbf{Un} \leq \tau$

[5] These conditions are more liberal than the corresponding conditions in Gordon and Jeffrey's type system. In their type system, for Public Challenge φ_1 (which corresponds to $\mathbf{N}_{\mathbf{Pub}}(\varphi_1, \emptyset)$ in our type system) to be tainted, φ_1 must also be empty.

respectively, where $\mathbf{Un}$ is the type of untrusted, non-secret data. Note that $\mathbf{DKey}$ is co-variant, $\mathbf{EKey}$ is contra-variant, and $\mathbf{SKey}$ is invariant; this is analogous to Pierce and Sangiorgi's IO types with subtyping [20].

$$\frac{\ell = \mathbf{Pub} \qquad \varphi_1 = \emptyset}{\mathbf{Pub}(\mathbf{N}_\ell(\varphi_1, \varphi_2))} \qquad \frac{\ell = \mathbf{Pub} \Rightarrow \varphi_2 = \emptyset}{\mathbf{Taint}(\mathbf{N}_\ell(\varphi_1, \varphi_2))} \qquad \frac{\mathbf{Pub}(\tau_1) \qquad \mathbf{Pub}(\tau_2)}{\mathbf{Pub}(\tau_1 \times \tau_2)}$$

$$\frac{\mathbf{Taint}(\tau_1) \qquad \mathbf{Taint}(\tau_2)}{\mathbf{Taint}(\tau_1 \times \tau_2)} \qquad \frac{\mathbf{Pub}(\tau) \qquad \mathbf{Taint}(\tau)}{\mathbf{Pub}(\mathbf{SKey}(\tau))} \qquad \frac{\mathbf{Pub}(\tau) \qquad \mathbf{Taint}(\tau)}{\mathbf{Taint}(\mathbf{SKey}(\tau))}$$

$$\frac{\mathbf{Taint}(\tau)}{\mathbf{Pub}(\mathbf{EKey}(\tau))} \qquad \frac{\mathbf{Pub}(\tau)}{\mathbf{Taint}(\mathbf{EKey}(\tau))} \qquad \frac{\mathbf{Pub}(\tau)}{\mathbf{Pub}(\mathbf{DKey}(\tau))} \qquad \frac{\mathbf{Taint}(\tau)}{\mathbf{Taint}(\mathbf{DKey}(\tau))}$$

Fig. 5. Predicates $\mathbf{Pub}$ and $\mathbf{Taint}$

Operations and relations on capabilities and types We write $dom(\varphi)$ for the set $\{A \mid \varphi(A) > 0\}$. We identify capabilities up to the following equality $\approx$:

$$\varphi_1 \approx \varphi_2 \iff (dom(\varphi_1) = dom(\varphi_2) \wedge \forall A \in dom(\varphi_1).\varphi_1(A) = \varphi_2(A)).$$

We write $\varphi \leq \varphi'$ if $\varphi(A) \leq \varphi'(A)$ holds for every $A \in dom(\varphi)$ and we define the summation of two capabilities by: $(\varphi_1 + \varphi_2)(A) = \varphi_1(A) + \varphi_2(A)$. This is a natural extension of the multiset union. We write $\varphi_1 - \varphi_2$ for the least φ such that $\varphi_1 \leq \varphi + \varphi_2$.

As we use indices to express dependent types, messages may be substituted in types. Let i be an index and M a message. The substitution $[M/i]\tau$ is defined inductively in the straight-forward manner, except for pair types where

$$[M/i](\tau_1 \times \tau_2) = ([M/i]\tau_1) \times ([M/(i+1)]\tau),$$

such that the index is shifted for the second component.

3.3 Typing

We introduce two forms of type judgments: $\Gamma; \varphi \vdash M : \tau$ for messages, and $\Gamma; \varphi \vdash P$ for processes, where Γ, called a type environment, is a sequence of type bindings of the form $x_1 : \tau_1, \ldots, x_n : \tau_n$. Judgment $\Gamma; \varphi \vdash M : \tau$ means that M evaluates to a value of type τ under the assumption that each name has the type described by Γ and that capability φ is available. $\Gamma; \varphi \vdash P$ means that P can be safely executed (i.e. without violation of correspondence assertions) if each free name has the type described by Γ and the capability φ is available. For example, $x : \mathbf{Un}; \{\mathbf{end}(x)\} \vdash \mathbf{end}\, x$ is valid but $x : \mathbf{Un}; \emptyset \vdash \mathbf{end}\, x$ is not.

We consider only the judgements that are *well-formed* in the sense that (i) φ refers to only the names bound in Γ, and (ii) Γ must be well-formed, i.e., if Γ is of the form $\Gamma_1, x : \tau, \Gamma_2$ then τ only refers to the names bound in Γ_1 and x is not bound in neither Γ_1 nor Γ_2. See [10] for the formal definition of the well-formedness of type environments and judgments. We freely permute bindings in type environments as long as they are well-formed; for example, we do not distinguish between $x : \mathbf{Un}, y : \mathbf{Un}$ and $y : \mathbf{Un}, x : \mathbf{Un}$.

Typing The typing rules are shown in Figure 6. The rule T-CAST says that the current capability can be used for discharging obligations and increasing capabilities of the name. T-CAST plays a role similar to the typing rule for cast processes in Gordon and Jeffrey's type system, but our cast is implicit and changes only the capabilities and obligations, not the shape of types. This difference is important for automated type inference. The other rules for messages are standard; T-PAIR is the standard rule for dependent sum types (except for the use of indices).

In the rules for processes, the capabilities shown by _ can be any capabilities. The rules are also similar to those of Gordon and Jeffrey, except for the rules T-OUT, T-IN, T-NEWN, and T-CHK. In rule T-OUT, we require that the type of message M_2 is public as it can be received by any process, including the attacker. Similarly, in rule T-IN we require that the type of the received value x is tainted, as it may come from any process. This is different from Gordon and Jeffrey's type system where the type of messages sent to or received from public channels must be $\mathbf{Un}$, and a subsumption rule allows any value of a public type to be typed as $\mathbf{Un}$ and a value of type $\mathbf{Un}$ to be typed as any tainted type. In effect, our type system can be considered a restriction of Gordon and Jeffrey's such that the subsumption rule is only allowed for messages sent or received via public channels. This point is important for automated type inference.

In rule T-NEWN, the obligation φ_1 is attached to the fresh name x and recorded in the atomic check capability. Capabilities corresponding to φ_1 can then later be extracted by a check operation if the obligation has been fulfilled. In rule T-CHK, $\mathbf{chk}_\ell(M_1, \varphi_4)$ in the conclusion means that the capability to check M_1 must be present. If the check succeeds, the capability φ_5 attached to M_2 can be extracted and used in P. In addition, the obligations attached to M_2 must be empty, i.e. all obligations initially attached to the name must have been fulfilled, and hence the capability φ_4 can be extracted and used in P. The above mechanism for extracting capabilities through obligations is different from Gordon and Jeffrey's type system in a subtle but important way, and provides more expressive power: see [10]. The remaining rules should be self-explanatory.

Example 2. Recall the POSH protocol in Figure 2. Let τ be $\mathbf{Un} \times \mathbf{N_{Pub}}(\emptyset, \{\mathbf{end}(0)\})$. Then the process describing the behavior of B ($net?n. \cdots$ in the last five lines) is typed as the upper part of Figure 7. Here, $\Gamma = \mathit{net} : \mathbf{Un}, \mathit{sk}_B : \mathbf{EKey}(\tau), n : \mathbf{Un}, \mathit{msg} : \mathbf{Un}$. Similarly, the part $\mathbf{decrypt}$ *ctext* $\mathbf{is}$ $\{\!|x|\!\}_{\mathit{pk}_B{}^{-1}}. \cdots$ of process A is typed as the lower part of Figure 7. Here, $\Gamma_2 = \mathit{net} : \mathbf{Un}, \mathit{pk}_B : \mathbf{DKey}(\tau), \mathit{non} : \mathbf{Un}, \mathit{ctext} : \mathbf{Un}$ and $\Gamma_3 = \Gamma_2, x : \tau, m : \mathbf{Un}, \mathit{non}' : \mathbf{N_{Pub}}(\emptyset, \{\mathbf{end}(m)\})$. Let P_1 be the entire process of the POSH protocol. It is typed by $\mathit{net} : \mathbf{Un}; \emptyset \vdash P_1$.

The SOPH and SOSH protocols in Figure 1 are typed in a similar manner. We show here only key types:

SOPH
$\mathit{pk}_B : \mathbf{EKey}(\mathbf{Un} \times \mathbf{N_{Pub}}(\{\mathbf{end}(0)\}, \emptyset)), \mathit{sk}_B : \mathbf{DKey}(\mathbf{Un} \times \mathbf{N_{Pub}}(\{\mathbf{end}(0)\}, \emptyset))$

$$\frac{}{\Gamma, x:\tau;\varphi \vdash x:\tau}\ \text{(T-VAR)} \qquad \frac{\Gamma;\varphi_1 \vdash M_1 : \tau_1 \qquad \Gamma;\varphi_2 \vdash M_2 : [M_1/0]\tau_2}{\Gamma;\varphi_1+\varphi_2 \vdash (M_1, M_2) : \tau_1 \times \tau_2}\ \text{(T-PAIR)}$$

$$\frac{\Gamma;\varphi_1 \vdash M_1 : \tau_1 \quad \Gamma;\varphi_2 \vdash M_2 : \mathbf{SKey}(\tau_1)}{\Gamma;\varphi_1+\varphi_2 \vdash \{M_1\}_{M_2} : \mathbf{N}_\ell(\emptyset,\emptyset)}\ \text{(T-SENC)} \qquad \frac{\Gamma;\varphi_1 \vdash M_1 : \tau \quad \Gamma;\varphi_2 \vdash M_2 : \mathbf{EKey}(\tau)}{\Gamma;\varphi_1+\varphi_2 \vdash \{\!|M_1|\!\}_{M_2} : \mathbf{N}_\ell(\emptyset,\emptyset)}\ \text{(T-AENC)}$$

$$\frac{\Gamma;\varphi_1 \vdash M : \mathbf{N}_\ell(\varphi_2,\varphi_3)}{\Gamma;\varphi_1+\varphi_2'+\varphi_3' \vdash M : \mathbf{N}_\ell(\varphi_2-\varphi_2', \varphi_3+\varphi_3')}\ \text{(T-CAST)}$$

$$\frac{}{\Gamma;\emptyset \vdash \mathbf{0}}\ \text{(T-ZERO)} \qquad \frac{\Gamma;\varphi_1 \vdash P_1 \qquad \Gamma;\varphi_2 \vdash P_2}{\Gamma;\varphi_1+\varphi_2 \vdash P_1 \,|\, P_2}\ \text{(T-PAR)} \qquad \frac{\Gamma;\emptyset \vdash P}{\Gamma;\emptyset \vdash *P}\ \text{(T-REP)} \qquad \frac{\Gamma;\varphi' \vdash P \qquad \varphi' \le \varphi}{\Gamma;\varphi \vdash P}\ \text{(T-CSUB)}$$

$$\frac{\begin{array}{c}\Gamma;\varphi_1 \vdash M_1 : \mathbf{N}_\ell(\emptyset,\emptyset)\\ \Gamma;\varphi_2 \vdash M_2 : \tau \qquad \mathbf{Pub}(\tau)\end{array}}{\Gamma;\varphi_1+\varphi_2 \vdash M_1!M_2}\ \text{(T-OUT)} \qquad \frac{\begin{array}{c}\Gamma;\varphi_1 \vdash M : \mathbf{N}_\ell(\emptyset,\emptyset)\\ \Gamma, x:\tau;\varphi_2 \vdash P \quad \mathbf{Taint}(\tau)\end{array}}{\Gamma;\varphi_1+\varphi_2 \vdash M?x.P}\ \text{(T-IN)} \qquad \frac{\Gamma, x:\mathbf{SKey}(\tau);\varphi \vdash P}{\Gamma;\varphi \vdash (\nu_{sym} x)P}\ \text{(T-NEWSK)}$$

$$\frac{\Gamma, x:\mathbf{N}_\ell(\varphi_1,\emptyset), \varphi + \{\mathbf{chk}_\ell(x,\varphi_1)\} \vdash P}{\Gamma;\varphi \vdash (\nu x)P}\ \text{(T-NEWN)} \qquad \frac{\Gamma, k_1:\mathbf{EKey}(\tau), k_2:\mathbf{DKey}(\tau);\varphi \vdash P}{\Gamma;\varphi \vdash (\nu_{asym} k_1, k_2)P}\ \text{(T-NEWAK)}$$

$$\frac{\Gamma;\varphi_1 \vdash M_1 : \mathbf{N}_\ell(\_,\_) \qquad \Gamma;\varphi_2 \vdash M_2 : \mathbf{SKey}(\tau) \qquad \Gamma, x:\tau;\varphi_3 \vdash P}{\Gamma;\varphi_1+\varphi_2+\varphi_3 \vdash \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P}\ \text{(T-SDEC)}$$

$$\frac{\Gamma;\varphi_1 \vdash M_1 : \mathbf{N}_\ell(\_,\_) \qquad \Gamma;\varphi_2 \vdash M_2 : \mathbf{DKey}(\tau) \qquad \Gamma, x:\tau;\varphi_3 \vdash P}{\Gamma;\varphi_1+\varphi_2+\varphi_3 \vdash \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P}\ \text{(T-ADEC)}$$

$$\frac{\Gamma;\varphi_1 \vdash M_1 : \mathbf{N}_\ell(\_,\_) \qquad \Gamma;\varphi_2 \vdash M_2 : \mathbf{N}_\ell(\emptyset,\varphi_5) \qquad \Gamma;\varphi_3+\varphi_4+\varphi_5 \vdash P}{\Gamma;\varphi_1+\varphi_2+\varphi_3+\{\mathbf{chk}_\ell(M_1,\varphi_4)\} \vdash \mathbf{check}\ M_1\ \mathbf{is}\ M_2.P}\ \text{(T-CHK)}$$

$$\frac{\Gamma;\varphi_1 \vdash M : \tau_1 \times \tau_2 \qquad \Gamma, y:\tau_1, z:[y/0]\tau_2;\varphi_2 \vdash P}{\Gamma;\varphi_1+\varphi_2 \vdash \mathbf{split}\ M\ \mathbf{is}\ (y,z).P}\ \text{(T-SPLIT)}$$

$$\frac{\Gamma;\varphi_1 \vdash M_1 : \tau_1 \times \tau_2 \qquad \Gamma;\varphi_2 \vdash M_2 : \tau_1 \qquad \Gamma, z:[M_2/0]\tau_2;\varphi_3 \vdash P}{\Gamma;\varphi_1+\varphi_2+\varphi_3 \vdash \mathbf{match}\ M_1\ \mathbf{is}\ (M_2,z).P}\ \text{(T-MATCH)}$$

$$\frac{\Gamma;\varphi+\{\mathbf{end}(M)\} \vdash P}{\Gamma;\varphi \vdash \mathbf{begin}\, M.P}\ \text{(T-BEGIN)} \qquad \frac{}{\Gamma;\varphi+\{\mathbf{end}(M)\} \vdash \mathbf{end}\, M}\ \text{(T-END)}$$

Fig. 6. Typing Rules

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma;\emptyset\vdash \mathit{msg}:\mathbf{Un}\quad \dfrac{\Gamma;\emptyset\vdash n:\mathbf{N_{Pub}}(\emptyset,\emptyset)}{\Gamma;\{\mathbf{end}(\mathit{msg})\}\vdash n:\mathbf{N_{Pub}}(\emptyset,\{\mathbf{end}(\mathit{msg})\})}}{\Gamma;\{\mathbf{end}(\mathit{msg})\}\vdash(\mathit{msg},n):\tau}}{\cdots}}{\Gamma;\{\mathbf{end}(\mathit{msg}),\mathbf{chk_{Pub}}(\mathit{msg},\emptyset)\}\vdash \mathit{net}!\{\!|(\mathit{msg},n)|\!\}_{\mathit{sk}_B}}}{\Gamma;\{\mathbf{chk_{Pub}}(\mathit{msg},\emptyset)\}\vdash \mathbf{begin}\,\mathit{msg}.\cdots}}{\mathit{net}:\mathbf{Un},\mathit{sk}_B:\mathbf{EKey}(\tau),n:\mathbf{Un};\emptyset\vdash(\nu \mathit{msg})\cdots}}{\mathit{net}:\mathbf{Un},\mathit{sk}_B:\mathbf{EKey}(\tau);\emptyset\vdash \mathit{net}?n.\cdots}$$

$$\dfrac{\dfrac{\dfrac{\Gamma_3;\{\mathbf{end}(m)\}\vdash \mathbf{end}\,m}{\Gamma_3;\{\mathbf{chk_{Pub}}(\mathit{non},\emptyset)\}\vdash \mathbf{check}\ \mathit{non}\ \mathbf{is}\ \mathit{non}'.\cdots}}{\Gamma_2,x:\tau;\{\mathbf{chk_{Pub}}(\mathit{non},\emptyset)\}\vdash \mathbf{split}\ x\ \mathbf{is}\ (m,\mathit{non}).\cdots}}{\Gamma_2;\{\mathbf{chk_{Pub}}(\mathit{non},\emptyset)\}\vdash \mathbf{decrypt}\ \mathit{ctext}\ \mathbf{is}\ \{\!|x|\!\}_{\mathit{pk}_B{}^{-1}}.\cdots}$$

Fig. 7. Partial Typing of the POSH Protocol

SOSH
$\mathit{pk}_A:\mathbf{EKey}(\mathbf{Un}\times\mathbf{N_{Pr}}(\emptyset,\{\mathbf{end}(0)\})), \mathit{sk}_A:\mathbf{DKey}(\mathbf{Un}\times\mathbf{N_{Pr}}(\emptyset,\{\mathbf{end}(0)\}))$
$\mathit{pk}_B:\mathbf{EKey}(\mathbf{Un}\times\mathbf{N_{Pr}}(\emptyset,\emptyset)),\quad \mathit{sk}_B:\mathbf{DKey}(\mathbf{Un}\times\mathbf{N_{Pr}}(\emptyset,\emptyset))$

Note that for POSH and SOPH the name qualifier must be $\mathbf{Pub}$, and only for the SOSH protocol may it be $\mathbf{Pr}$. □

3.4 Soundness of the Type System

The soundness of the type system is stated as follows.

Theorem 1 (soundness). *If $x_1:\mathbf{Un},\ldots,x_m:\mathbf{Un};\emptyset\vdash P$, then P is robustly safe.*

To prove this theorem above we first prepare the following lemma, which implies that, in the definition of robust safety, it is sufficient to consider only well-typed opponent processes.

Lemma 1. *If O is a process that contains no begin/end/check, then there exists O' that satisfies the following conditions:*

1. *$x_1:\mathbf{Un},\ldots,x_m:\mathbf{Un};\emptyset\vdash O'$, where $\{x_1,\ldots,x_k\}=\mathbf{FN}(O)$.*
2. *For any process P, if $P\,|\,O'$ is safe then so is $P\,|\,O$.*

Proof Sketch Let y be a name not occurring in O and let O' be the process obtained from O by replacing any occurrence of $M_1!M_2$ and $M?x.P$ with $(\nu y)(y!M_1\,|\,y?z.z!M_2)$ and $(\nu y)(y!M\,|\,y?z.z?x.P)$, respectively. In this way we are free to change the types of opponent values like $\mathbf{Un}$, $\mathbf{Un}\times\mathbf{Un}$, and $\mathbf{DKey}(\mathbf{Un})$ by communicating them through channels of public types. Then, O' satisfies the required properties. See Appendix C for more details. □

By the lemma above, to prove Theorem 1, it suffices to show the following lemma.

Lemma 2. *If $\emptyset; \emptyset \vdash P$, then P is safe.*

Proof. See Appendix D.

Returning to the proof of the soundness theorem we then have:

Proof of Theorem 1 Suppose $x_1 : \mathbf{Un}, \ldots, x_n : \mathbf{Un}; \emptyset \vdash P$. Let O be a process that does not contain begin/end/check. We need to show that $P \mid O$ is safe. By Lemma 1, there exists a process O' such that (i) $y_1 : \mathbf{Un}, \ldots, y_k : \mathbf{Un}; \emptyset \vdash O'$ and (ii) if $P \mid O'$ is safe, so is $P \mid O$. Let $\{z_1 : \mathbf{Un}, \ldots, z_m : \mathbf{Un}\} = \{x_1 : \mathbf{Un}, \ldots, x_n : \mathbf{Un}\} \cup \{y_1 : \mathbf{Un}, \ldots, y_k : \mathbf{Un}\}$. Then, by weakening and the typing rules, we have $\emptyset; \emptyset \vdash (\nu z_1) \cdots (\nu z_m)(P \mid O')$. By Lemma 2, $(\nu z_1) \cdots (\nu z_m)(P \mid O')$ is safe. By the definition of the safety, $P \mid O'$ is also safe. By condition (ii) above, $P \mid O$ is also safe. □

4 Type Inference

We now briefly discuss type inference. For this we impose a minor restriction to the type system, namely that in rule T-PAIR, if M_1 is not a name then the indice 0 cannot occur in τ_2. Similarly, in rule T-MATCH we require that index 0 does not occur unless M_2 is a name. These restrictions prevent the size of types and capabilities from blowing up. Given as input a process P with free names $x_1, \ldots, x_n$, the algorithm to decide $x_1 : \mathbf{Un}, \ldots, x_n : \mathbf{Un}; \emptyset \vdash P$ proceeds as follows:

1. Determine the *shape of the type* (or simple type) of each term via a standard unification algorithm, and construct a template of a type derivation tree by introducing qualifier and capability variables.
2. Generate a set C of constraints on qualifier and capability variables based on the typing rules such that C is satisfiable if and only if $x_1 : \mathbf{Un}, \ldots, x_n : \mathbf{Un}; \emptyset \vdash P$.
3. Solve the qualifier constraints.
4. Transform the capability constraints to linear inequalities over the rational numbers.
5. Use linear programming to determine if the linear inequalities are satisfiable.

In step 1, we can assume that there are no consecutive applications of T-CAST and T-CSUB. Thus, the template of a type derivation tree can be uniquely determined: for each process and message constructor there is an application of the rule matching the constructor followed by at most one application of T-CAST or T-CSUB.

At step 3 we have a set of constraints C of the form:

$$\{\ell_i = \ell'_i \mid i \in I\} \cup \{(\ell''_j = \mathbf{Pub}) \Rightarrow (\varphi_j = \emptyset) \mid j \in J\} \cup C_1$$

where I and J are finite sets, $\ell_i, \ell'_i, \ell''_j$ are qualifier variables or constants, and C_1 is a set of effect constraints (like $\varphi_1 \leq \varphi_2$). Here, constraints on qualifiers come from equality constraints on types and conditions $\mathbf{Pub}(\tau)$ and $\mathbf{Taint}(\tau)$. In particular, $(\ell''_j = \mathbf{Pub}) \Rightarrow (\varphi_j = \emptyset)$ comes from the rule for $\mathbf{Taint}(\mathbf{N}_{\ell''_j}(\varphi, \varphi_j))$. By obtaining the most general unifier θ of the first set of constraints $\{\ell_i = \ell'_i \mid i \in I\}$ we obtain the constraint set $C' \equiv \{(\theta\ell''_j = \mathbf{Pub}) \Rightarrow (\theta\varphi_j = \emptyset) \mid j \in J\} \cup \theta C_1$. Let $\gamma_1, \ldots, \gamma_k$ be the remaining qualifier variables, and let $\theta' = [\mathbf{Pr}/\gamma_1, \ldots, \mathbf{Pr}/\gamma_k]$. Then C is satisfiable if and only if $\theta' C'$ is satisfiable. Thus, we obtain the set $\theta' C'$ of effect constraints that is satisfiable if and only if $x_1 : \mathbf{Un}, \ldots, x_n : \mathbf{Un}; \emptyset \vdash P$ holds.

Except for step 3, the above algorithm is almost the same as our previous work and we refer the interested reader to [18, 19]. By a similar argument to that given in [19] we can show that under the assumptions that the size of each begin/end assertion occurring in the protocol is bounded by a constant and that the size of simple types is polynomial in the size of the protocol, the type inference algorithm runs in polynomial time.

Example 3. Recall the POSH protocol in Figure 2. By the simple type inference in step 1 we get the following types for names:

$$\mathit{non}, \mathit{non}' : \mathbf{N}, \mathit{pk}_B : \mathbf{DKey}(\mathbf{N} \times \mathbf{N}), \ldots$$

By preparing qualifier and capability variables we get the following elaborated types and constraints on those variables:

$$\begin{array}{l} \mathit{non} : \mathbf{N}_{\gamma_1}(\xi_{0,o}, \xi_{0,c}), \mathit{non}' : \mathbf{N}_{\gamma_1'}(\xi'_{0,o}, \xi'_{0,c}), \ldots \\ \mathbf{Pub}(\mathbf{N}_{\gamma_1}(\xi_{0,o}, \xi_{0,c})) \quad \gamma_1 = \gamma_1' \quad \xi_6 \leq \xi_3 + \xi_4 + \xi_5 \\ \xi_2 \geq \xi'_{0,o} + (\xi_5 - \xi'_{0,c}) \quad \xi_7 \geq \xi_1 + \xi_2 + \xi_3 + \{\mathbf{chk}_{\gamma_1}(\mathit{non}, \xi_4)\} \quad \cdots \end{array}$$

Here, the constraint $\mathbf{Pub}(\mathbf{N}_{\gamma_1}(\xi_{0,o}, \xi_{0,c}))$ comes from $\mathit{net}!\mathit{non}$, and the other constraints from **check** *non* **is** *non*. $\cdots$. By solving the qualifier constraints, we get $\gamma_1 = \gamma_1' = \mathbf{Pub}, \ldots$, and are left with constraints on capability variables. By computing (an over-approximation of) the domain of each capability, we can reduce it to constraints on linear inequalities. For example, by letting $\xi_i = \{\mathbf{chk}_{\mathbf{Pub}}(\mathit{non}, \xi_4) \mapsto x_i, \mathbf{end}(m) \mapsto y_i, \ldots\}$, the last constraint is reduced to:

$$x_7 \geq x_1 + x_2 + x_3 + 1 \quad y_7 \geq y_1 + y_2 + y_3 + 0 \quad \cdots$$

5 Implementation and Experiments

We have implemented a protocol verifier SPICA2 based on the type system and inference algorithm discussed above. The implementation is mostly based on the formalization in the paper, except for a few extensions such as sum types and private channels to securely distribute initial keys. The implementation can be tested at `http://www.kb.ecei.tohoku.ac.jp/~koba/spica2/`.

We have tested SPICA2 on several protocols with the results of the experiments shown in Table 5. Experiments were conducted using a machine with a 3GHz CPU and 2GB of memory.

The descriptions of the protocols used in the experiments are available at the above URL. `POSH`, `SOPH`, and `SOSH` are (spi$_{CA}$-notations of) the protocols given in Figure 1. `GNSL` is the generalized Needham-Schroeder-Lowe protocol [9]: see Appendix B.2 for details. `Otway-Ree` is Otway-Ree protocol using symmetric keys. `Iso-two-pass` is from [16], and the remaining protocols are the Needham-Schroeder-Lowe protocol and its variants, taken from the sample programs of Cryptyc [17] (but with type annotations and casts removed). `ns-flawed` is the original flawed version, `nsl-3` and `nsl-7` are 3- and 7-message versions of Lowe's fix, respectively. See [17] for the other three. As the table shows, all the protocols have been correctly verified or rejected. Furthermore, verification succeeded in less than a second except for `GNSL`. For `GNSL`, the slow-down is caused by the explosion of the number of atomic capabilities to be considered, which blows up the number of linear inequalities obtained from capability constraints.

Protocols	Typing	Time (sec.)	Protocols	Typing	Time (sec.)
POSH	yes	0.001	ns-flawed	no	0.007
SOPH	yes	0.001	nsl-3	yes	0.015
SOSH	yes	0.001	nsl-7	yes	0.049
GNSL	yes	7.40	nsl-optimized	yes	0.012
Otway-Ree	yes	0.019	nsl-with-secret	yes	0.023
Iso-two-pass	yes	0.004	nsl-with-secret-optimized	yes	0.016

Table 1. Experimental results

6 Extensions

In this section, we hint on how to modify our type system and type inference algorithm to deal with other features. Formalization and implementation of the extensions are left for future work.

Our type system can be easily adopted to deal with non-injective correspondence [14], which allows multiple end-events to be matched by a single begin-event. It suffices to relax the typing rules, for example, by changing the rules for begin- and end-events to:

$$\frac{\Gamma; \varphi + \{\mathbf{end}(M) \mapsto r\} \vdash P \qquad r > 0}{\Gamma; \varphi \vdash \mathbf{begin}\, M.P} \qquad \frac{r > 0}{\Gamma; \varphi + \{\mathbf{end}(M) \mapsto r\} \vdash \mathbf{end}\, M}$$

The capabilities attached to a name can now be extracted without using the check operation:

$$\frac{\Gamma\ \varphi \vdash M : \mathbf{N}_\ell(\varphi_1, \varphi_2)}{\Gamma\ \varphi + \varphi_2 \vdash M : \mathbf{N}_\ell(\varphi_1, \varphi_2)}$$

Fournet et al. [12] generalized begin- and end-events by allowing predicates to be defined by Datalog programs. For example, the process:

$$\textbf{assume}\ \textit{employee}(a); \textbf{expect}\ \textit{canRead(a, handbook)}$$

is safe in the presence of the clause "canRead(X,handbook) :- employee(X)". Here, the primitives **assume** and **expect** are like non-injective versions of **begin** and **end**. A similar type system can be obtained by extending our capabilities to mappings from ground atomic formulas to rational numbers (where $\varphi(L) > 0$ means L holds), and introducing rules for assume and expect similar to the rules above for begin and end-events. To handle clauses like "canRead(X,handbook) :- employee(X)", we can add the following rule:

$$\frac{\Gamma; \varphi + \{L \mapsto r\} \vdash P \qquad \text{There is an (instance of) clause } L :- L_1, \ldots, L_k \quad r \leq \varphi(L_i) \text{ for each } i \in \{1, \ldots, k\}}{\Gamma; \varphi \vdash P}$$

This allows us to derive a capability for L whenever there are capabilities for $L_1, \ldots, L_k$. To reduce capability constraints to linear programming problems, it suffices to extend the algorithm to obtain the domain of each effect [19], taking clauses into account (more precisely, if there is a clause $L :- L_1, \ldots, L_k$ and $\theta L_1, \ldots, \theta L_k$ are in the domain of φ, we add θL to the domain of φ).

To deal with trust and witness in [16], we need to mix type environments and capabilities, so that type environments can also be attached to names and passed around. The resulting type system is rather complex, so that we leave the details to another paper.

7 Related Work

The present work extends two lines of previous work: Gordon and Jeffrey's type systems for authenticity [15, 16], and Kikuchi and Kobayashi's work to enable type inference for symmetric cryptographic protocols [19]. In our opinion the extension is non-trivial, requiring the generalization of name types and a redesign of the type system. This has yielded a fully-automated and efficient protocol verifier. As for the expressive power, the fragment of Gordon and Jeffrey's type system (subject to minor restrictions) without trust and witness can be easily embedded into our type system. On the other hand, thanks to the uniform treatment of name types in terms of capabilities and obligations, our type system can express protocols that are not typable in Gordon and Jeffrey's type system, like the GNSL multi-party protocol [9]. See [10] for more details.

Gordon et al. [3, 4] extended their work to verify source code-level implementation of cryptographic protocols by using refinement types. Their type systems still require refinement type annotations. We plan to extend the ideas of the present work to enable partial type inference for their type system. Bugliesi, Focardi, and Maffei [6, 11, 7] have proposed a protocol verification method that is closely related to Gordon and Jeffrey's type systems. They [11] developed an algorithm for automatically inferring *tags* (which roughly correspond to Gordon and Jeffrey's types in [15, 16]). Their inference algorithm is based on exhaustive search of taggings by backtracking, hence our type inference would be more efficient. As in Gordon and Jeffrey type system, their tagging and typing system is specialized for the typical usage of nonces in two-party protocols, and appears to be inapplicable to multi-party protocols like GNSL.

There are automated protocol verification tools based on other approaches, such as ProVerif [5] and Scyther [8]. Advantages of our type-based approach are: (i) it allows modular verification of protocols[6]; (ii) it sets up a basis for studies of partial or full

[6] Although the current implementation of SpiCA2 only supports whole protocol analysis, it is easy to extend it to support partial type annotations to enable modular verification. For that purpose, it suffices to allow bound variables to be annotated with types, and generate the corresponding constraints during type inference. For example, for a type-annotated input $M?(x : \tau_1).P$, we just need to add the subtype constraint $\tau_1 \leq \tau$ to rule T-In.

type inference for more advanced type systems for protocol verification [4] (for an evidence, recall Section 6); and (iii) upon successful verification, it generates types as a certificate, which explains why the protocol is safe, and can be independently checked by other type-based verifiers [16, 4]. On the other hand, ProVerif [5] and Scyther [8] have an advantage that they can generate an attack scenario given a flawed protocol. Thus, we think that our type-based tool is complementary to existing tools.

8 Conclusion

We have redesigned Gordon and Jeffrey's type system for authenticity of asymmetric cryptographic protocols, and developed a type inference algorithm. This has enabled fully automated type-based protocol verification, which requires no type annotations. Future work includes an extension to deal with trust and witness in Gordon and Jeffrey's type system.

References

1. M. Abadi. Secrecy by typing in security protocols. *JACM*, 46(5):749–786, 1999.
2. M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.
3. J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 17–32, 2008.
4. K. Bhargavan, C. Fournet, and A. D. Gordon. Modular verification of security protocol code by typing. In *Proceedings of POPL 2010*, pages 445–456, 2010.
5. B. Blanchet. From Secrecy to Authenticity in Security Protocols. In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *LNCS*, pages 342–359. Springer-Verlag, 2002.
6. M. Bugliesi, R. Focardi, and M. Maffei. Analysis of typed analyses of authentication protocols. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005)*, pages 112–125, 2005.
7. M. Bugliesi, R. Focardi, and M. Maffei. Dynamic types for authentication. *Journal of Computer Security*, 15(6):563–617, 2007.
8. C. J. F. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *Proceedings of ACM Conference on Computer and Communications Security (CCS 2008)*, pages 119–128, 2008.
9. C. J. F. Cremers and S. Mauw. A family of multi-party authentication protocols - extended abstract. In *Proceedings of WISSEC'06*, 2006.
10. M. Dahl, N. Kobayashi, Y. Sun, and H. Hüttel. Type-based automated verification of authenticity in asymmetric cryptographic protocols. Full version, available at `http://www.kb.ecei.tohoku.ac.jp/~koba/papers/protocol-full.pdf`, 2011.
11. R. Focardi, M. Maffei, and F. Placella. Inferring authentication tags. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS 2005)*, pages 41–49, 2005.
12. C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization policies. *ACM Trans. Prog. Lang. Syst.*, 29(5), 2007.
13. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. Technical Report MRS-TR-2002-31, Microsoft Research, Aug. 2002.

14. A. D. Gordon and A. Jeffrey. Typing one-to-one and one-to-many correspondences in security protocols. In *Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium (ISSS 2002)*, volume 2609 of *LNCS*, pages 263–282. Springer-Verlag, 2002.
15. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of Computer Security*, 11(4):451–520, 2003.
16. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3-4):435–483, 2004.
17. C. Haack and A. Jeffrey. Cryptyc. `http://www.cryptyc.org/`, 2004.
18. D. Kikuchi and N. Kobayashi. Type-based verification of correspondence assertions for communication protocols. In *Proceedings of APLAS 2007*, volume 4807 of *LNCS*, pages 191–205. Springer-Verlag, 2007.
19. D. Kikuchi and N. Kobayashi. Type-based automated verification of authenticity in cryptographic protocols. In *Proceedings of ESOP 2009*, volume 5502 of *LNCS*, pages 222–236. Springer-Verlag, 2009.
20. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
21. T. Y. Woo and S. S. Lam. A semantic model for authentication protocols. In *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–193, 1993.

A Well-formedness of Type Environments and Judgments

The well-formedness conditions for type judgments and type environments are given in Figure 8. Here, $\uparrow N$ denotes the set of extended names obtained from N by replacing each number i in N with $i+1$. For example, $\uparrow\{x,y,0\} = \{x,y,1\}$.

$$\frac{\vdash_{\mathtt{wf}} \Gamma \qquad \mathbf{FEN}(\varphi) \subseteq dom(\Gamma) \qquad dom(\Gamma) \vdash_{\mathtt{wf}} \tau}{\vdash_{\mathtt{wf}} \Gamma; \varphi \vdash M : \tau} \quad \text{(WFJ-MESSAGE)}$$

$$\frac{\vdash_{\mathtt{wf}} \Gamma \qquad \mathbf{FEN}(\varphi) \subseteq dom(\Gamma)}{\vdash_{\mathtt{wf}} \Gamma; \varphi \vdash P} \quad \text{(WFJ-PROC)}$$

$$\frac{}{\vdash_{\mathtt{wf}} \emptyset} \quad \text{(WFTE-EMPTY)}$$

$$\frac{\vdash_{\mathtt{wf}} \Gamma \qquad dom(\Gamma) \vdash_{\mathtt{wf}} \tau \qquad x \notin dom(\Gamma)}{\vdash_{\mathtt{wf}} \Gamma, x : \tau} \quad \text{(WFTE-EXT)}$$

$$\frac{\mathbf{FEN}(\varphi_1) \cup \mathbf{FEN}(\varphi_2) \subseteq N}{N \vdash_{\mathtt{wf}} \mathbf{N}_\ell(\varphi_1, \varphi_2)} \quad \text{(WFT-NAME)}$$

$$\frac{N \vdash_{\mathtt{wf}} \tau}{N \vdash_{\mathtt{wf}} \mathbf{SKey}(\tau)} \quad \text{(WFT-SKEY)}$$

$$\frac{N \vdash_{\mathtt{wf}} \tau}{N \vdash_{\mathtt{wf}} \mathbf{DKey}(\tau)} \quad \text{(WFT-DKEY)}$$

$$\frac{N \vdash_{\mathtt{wf}} \tau}{N \vdash_{\mathtt{wf}} \mathbf{EKey}(\tau)} \quad \text{(WFT-EKEY)}$$

$$\frac{N \vdash_{\mathtt{wf}} \tau_1 \qquad \{0\} \cup \uparrow N \vdash_{\mathtt{wf}} \tau_2}{N \vdash_{\mathtt{wf}} \tau_1 \times \tau_2} \quad \text{(WFT-PAIR)}$$

Fig. 8. Well-formedness conditions for type judgments

B Relation to Gordon-Jeffrey Type System

We now turn to the subject of relating our type system to that of Gordon and Jeffrey. There are two main points here. First, we show that the fragment of the Gordon-Jeffrey type system without witness and trust can be embedded into our type system. We make some additional restrictions regarding nonce types but these appear to be without loss of expressive power for practice purposes. Second, we show that our formulation of nonce types actually allows us to type realistic protocols untypable in the Gordon-Jeffrey type system.

B.1 Partial Embedding of Gordon and Jeffrey's Type System

Restrictions In order to show an embedding of their type system into ours we have to make a few modifications. Most notably, we (i) leave out an embedding for witness and trust processes, (ii) inline the message subsumption rule, (iii) modify check atomic effects to additionally contain an effect es, and (iv) change the typing of processes dealing with nonce types.

Modification (ii) means that the subsumption rule for messages is removed and inlined in rules PROC OUTPUT UN and PROC INPUT UN (and similarly for PROC REPEAT INPUT UN) instead:

$$\frac{\Gamma \vdash_{\mathbf{GJ}} M : \mathsf{Un} \qquad \Gamma \vdash_{\mathbf{GJ}} N : T \qquad T \leq_{\mathbf{GJ}} \mathsf{Un}}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{out}\ M\ N : []} \quad \text{(PROC OUTPUT UN)}$$

$$\frac{\Gamma \vdash_{\mathbf{GJ}} M : \mathsf{Un} \qquad \Gamma, y : T \vdash_{\mathbf{GJ}} P : es \qquad \mathsf{Un} \leq_{\mathbf{GJ}} T}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{inp}\ M\ (y : T); P : es} \quad \text{(PROC INPUT UN)}$$

This modification is justified by the belief that honest processes should not have to apply subsumptions in more general ways than this, in that doing so means changing a type from or to something else than Un.

Modifications (iii) and (iv) mean that typing rules PROC CHALLENGE and PROC CHECK are changed as follows:

$$\frac{\Gamma \vdash_{\mathbf{GJ}} fs \qquad \Gamma, x : l\ \mathsf{Challenge}\ fs \vdash_{\mathbf{GJ}} P : es}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{new}\ (x : l\ \mathsf{Challenge}\ fs); P : es - [\mathsf{check}\ l\ x\ fs]} \quad \text{(PROC CHALLENGE)}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathbf{GJ}} M : l\ \mathsf{Challenge}\ es_C \\ \Gamma \vdash_{\mathbf{GJ}} N : l\ \mathsf{Response}\ es_R \qquad \Gamma \vdash_{\mathbf{GJ}} P : fs \\ es = fs - (es_C + es_R) \qquad es'_C = es_C\end{array}}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{check}\ M\ \mathsf{is}\ N; P : es + [\mathsf{check}\ l\ M\ es'_C]} \quad \text{(PROC CHECK)}$$

One consequence of this is that M in rule PROC CHECK can no longer be bound to names with different check capabilities. Moreover, while the addition of condition $es'_C = es_C$ makes rule PROC CHECK more restrictive than in the original formulation, breaking this condition does require use of either subtyping or matching in a way that respectively should not be done by honest processes, or does not appear to be required by a significant number of protocols. In the former case, subtyping must be used to turn a public nonce into a private nonce. In the latter case, match can be used to turn a check capability for one name into a check capability for another name. This however, seems to be possible only for protocols that deadlock.

We apply a few less important modifications to the type system as well. Type Top is removed and typing rules PROC BEGIN and PROC END are modified to simply require M to be of some type T instead:

$$\frac{\Gamma \vdash_{\mathbf{GJ}} M : T \qquad \Gamma \vdash_{\mathbf{GJ}} P : es}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{begin}\ M; P : es - [\mathsf{end}\ M]} \quad \text{(PROC BEGIN)}$$

$$\frac{\Gamma \vdash_{\mathbf{GJ}} M : T \qquad \Gamma \vdash_{\mathbf{GJ}} P : es}{\Gamma \vdash_{\mathbf{GJ}} \mathsf{end}\ M; P : es + [\mathsf{end}\ M]} \qquad \text{(PROC END)}$$

As far as we checked, all the protocols (without trust and witness) typed in [16] are typable under all the modifications above.

For the calculus we first consider a variant of the process calculus without $\mathsf{inl}\ (M)$ and $\mathsf{inr}\ (M)$ messages, and without case processes. Secondly, we restrict end processes to match our non-continuous variant. Thirdly, we restrict the generation of key pairs so that messages $\mathsf{Encrypt}\ (M)$ and $\mathsf{Decrypt}\ (M)$ may only occur immediately following the creation of a key pair (see below). The restrictions on the calculus can be removed by an easy extension of our calculus and type system, and are improsed here just for the sake of simplicity.

Embedding To ease the presentation we first add a derived process to both calculi

$$\mathbf{let}\ y\ \mathbf{is}\ x\ \mathbf{in}\ P = \mathbf{match}\ (c, x)\ \mathbf{is}\ (c, y).P$$

for some constant c, along with typing rule

$$\frac{\Gamma; \varphi_1 \vdash x : \tau \qquad \Gamma, y : \tau; \varphi_2 \vdash P}{\Gamma; \varphi_1 + \varphi_2 \vdash \mathbf{let}\ y\ \mathbf{is}\ x\ \mathbf{in}\ P} \qquad \text{(T-LET)}$$

derivable from typing rules T-MATCH and T-PAIR.

$$\begin{aligned}
[(x{:}T, U)] &= [T] \times [0/x][U] \\
[\mathsf{Un}] &= \mathbf{N_{Pub}}(\emptyset, \emptyset) \\
[\mathsf{SharedKey}(T)] &= \mathbf{SKey}([T]) \\
[\mathsf{Public\ Challenge}\ es] &= \mathbf{N_{Pub}}([es], \emptyset) \\
[\mathsf{Private\ Challenge}\ es] &= \mathbf{N_{Pr}}([es], \emptyset) \\
[\mathsf{Public\ Responce}\ es] &= \mathbf{N_{Pub}}(\emptyset, [es]) \\
[\mathsf{Private\ Responce}\ es] &= \mathbf{N_{Pr}}(\emptyset, [es]) \\
[\mathsf{Encrypt\ Key}(T)] &= \mathbf{EKey}([T]) \\
[\mathsf{Decrypt\ Key}(T)] &= \mathbf{DKey}([T])
\end{aligned}$$

Fig. 9. Type mapping

The central ingredient in the embedding is the mapping of types. For this we first have a straightforward mapping of messages M and effects es relying purely on syntactical conversion; for this reason we shall often simply write M instead of $[M]$ and es instead of $[es]$. We then define the mapping $[T]$ of a Gordon-Jeffrey type T as in Figure 9. We extend this to environments Γ in the point-wise manner. Note that as discussed above we consider a variant of the Gordon-Jeffrey type system withtop Top and Sum types and our mapping is left undefined for these. As we furthermore allow only a restricted use of KeyPair types the mapping is also left undefined for these as well as for CR types since these should not occur in user code.

In the mapping of processes (Figure 10) we use the provided typing information in the case of name restriction. As discussed above we impose some restrictions on processes and the mapping is left undefined for these; for the remaining processes the mapping is defined recursively.

$$
\begin{aligned}
[\mathsf{cast}\ x\ \mathsf{is}\ (y\!:\!T); P] &= \mathbf{let}\ y\ \mathbf{is}\ x\ \mathbf{in}\ [P] \\
[\mathsf{check}\ x\ \mathsf{is}\ (y\!:\!T); P] &= \mathbf{check}\ x\ \mathbf{is}\ y.[P] \\
[\mathsf{end}\ M; 0] &= \mathbf{end}\ M \\
[\mathsf{begin}\ M; P] &= \mathbf{begin}\ M.[P] \\
[\mathsf{new}\ (x\!:\!\mathsf{Un}); P] &= (\nu x)[P] \\
[\mathsf{new}\ (x\!:\!l\ \mathsf{Challenge}\ es); P] &= (\nu x)[P] \\
[\mathsf{new}\ (x\!:\!\mathsf{SharedKey}(T)); P] &= (\nu_{sym} x)[P] \\
\begin{bmatrix} \mathsf{new}\ (x\!:\!\mathsf{KeyPair}(T)); \\ \mathsf{let}\ y\ \mathsf{is}\ \mathsf{Encrypt}(x)\ \mathsf{in} \\ \mathsf{let}\ z\ \mathsf{is}\ \mathsf{Decrypt}(x)\ \mathsf{in} \\ P \quad (x \notin \mathbf{FN}(P)) \end{bmatrix} &= (\nu_{asym} y, z)[P] \\
[\mathsf{new}\ (x\!:\!\mathsf{Un}); P] &= (\nu x)[P] \\
[\mathsf{out}\ M\ N] &= M!N \\
[\mathsf{inp}\ M\ (x\!:\!T); P] &= M?x.[P] \\
[\mathsf{repeat}\ \mathsf{inp}\ M\ (x\!:\!T); P] &= *M?x.[P] \\
[\mathsf{split}\ M\ \mathsf{is}\ (x\!:\!T, y\!:\!U); P] &= \mathbf{split}\ M\ \mathbf{is}\ (x, y).[P] \\
[\mathsf{match}\ M\ \mathsf{is}\ (N, y\!:U); P] &= \mathbf{match}\ M\ \mathbf{is}\ (N, y).[P] \\
[\mathsf{decrypt}\ M\ \mathsf{is}\ \{x\!:\!T\}_N; P] &= \mathbf{decrypt}\ M\ \mathbf{is}\ \{x\}_N.[P] \\
[\mathsf{decrypt}\ M\ \mathsf{is}\ \{\!|x\!:\!T|\!\}_{N^{-1}}; P] &= \mathbf{decrypt}\ M\ \mathbf{is}\ \{\!|x|\!\}_{N^{-1}}.[P] \\
[P \mid Q] &= [P] \mid [Q] \\
[\mathsf{stop}] &= \mathbf{0}
\end{aligned}
$$

Fig. 10. Process mapping

Theorem 2. *If $\Gamma \vdash_{\mathbf{GJ}} P : es$ then $[\Gamma]; [es] \vdash [P]$.*

B.2 Limitations of Gordon and Jeffrey Type System

The converse of the result of the previous subsection does not hold, i.e. there are some realistic protocols that are typable in our type system but not in the Gordon-Jeffrey type system. This is a consequence of how nonces are typed: in their type system, nonce types are given two kinds of types: ℓ Challenge es and ℓ Response es. This forces each nonce to be used in at most two phases, first as a challenge, and then as a response. Our name types do not impose such restriction. The rest of this section illustrates two cases of protocols typable in our type system but not in Gordon and Jeffrey's type system.

Generalised Needham-Schroeder-Lowe The GNSL multi-party protocol [9] establishes mutual authentication between p parties using a minimal number of messages. For $p = 3$ with participants named R_0, R_1, and R_2, the protocol looks as follows:

```
R0 -> R1: {|R0,R2,n0|}pk1
R1 -> R2: {|R0,R1,n0,n1|}pk2
R2 -> R0: {|R1,R2,n0,n1,n2|}pk0
R0 -> R1: {|n1,n2|}pk1
R1 -> R2: {|n2|}pk2
```

where n_i is a nonce generated by R_i and pk_i the public key of a key pair belonging to R_i.

Participant R_0 first sends his nonce n_0 to R_1 who appends his nonce n_1 before forwarding to R_2. Likewise, R_2 appends his nonce n_2 before sending all nonces back to R_0. For the second round, R_0 checks his nonce against the one received from R_2 and sends n_1 and n_2 to R_1. After checking his nonce, R_1 sends n_2 to R_2 who then also checks his nonce.

The authenticity property dictates that each party agrees with both of the other parties on who the participants are, and is specified like so:

```
R0 -> R1: {|R0,R2,n0|}pk1
R1 begins (R0,R1,R2,01)
R1 begins (R0,R1,R2,21)
R1 -> R2: {|R0,R1,n0,n1|}pk2
R2 begins (R0,R1,R2,02)
R2 begins (R0,R1,R2,12)
R2 -> R0: {|R1,R2,n0,n1,n2|}pk0
R0 ends   (R0,R1,R2,01)
R0 ends   (R0,R1,R2,02)
R0 begins (R0,R1,R2,10)
R0 begins (R0,R1,R2,20)
R0 -> R1: {|n1,n2|}pk1
R1 ends   (R0,R1,R2,10)
R1 ends   (R0,R1,R2,12)
R1 -> R2: {|n2|}pk2
R2 ends   (R0,R1,R2,20)
R2 ends   (R0,R1,R2,21)
```

for some constants $01, \ldots, 21$. Note that for this property to hold we must assume that none of the parties R_0, R_1, and R_2 are compromised.

From the type system's point of view, the authenticity property e.g. means that a end-capability from both R_1 and R_2 must be transferred to R_0 using one nonce n_0. This is a problem for Gordon and Jeffrey's type system since capabilities can only be attached to nonces once due to the fact that the PROC CAST typing rule will only accept a Challenge type and additionally turn it into a Response type. Our type system does not have this limitation and can type the protocol with the following initial types for the nonces:

$$\begin{array}{l} n_0 : \mathbf{N_{Pr}}(\{\mathbf{end}(\ldots, 01), \mathbf{end}(\ldots, 21), \mathbf{end}(\ldots, 02)\}, \emptyset) \\ n_1 : \mathbf{N_{Pr}}(\{\mathbf{end}(\ldots, 10), \mathbf{end}(\ldots, 12)\}, \emptyset) \\ n_2 : \mathbf{N_{Pr}}(\{\mathbf{end}(\ldots, 20), \mathbf{end}(\ldots, 21)\}, \emptyset) \end{array}$$

so that the type of n_0 is later changed by R_1 to $\mathbf{N_{Pr}}(\{\mathbf{end}(\ldots, 02)\}, \emptyset)$ and then by R_2 to $\mathbf{N_{Pr}}(\emptyset, \emptyset)$. When n_0 makes it back to R_0 it can extract capabilities $\{\mathbf{end}(\ldots, 01), \mathbf{end}(\ldots, 21), \mathbf{end}(\ldots, 02)\}$ and use $\mathbf{end}(\ldots, 21)$ to discharge the obligation attached to n_2. These changes of name types cannot be expressed in Gordon and Jeffrey's type system.

SOPH Handshakes Another example of a protocol that is typable in our type system but not in Gordon and Jeffrey's is the SOPH handshake protocol in Figure 1. As mentioned in Example 2 in Section 3.3, *pk*$_B$ should have type $\mathbf{EKey}(\mathbf{Un} \times \mathbf{N_{Pub}}(\{\mathbf{end}(0) \mapsto 1\}, \emptyset))$, which corresponds to $\mathsf{Encrypt\ Key}(x : \mathsf{Un}, \mathsf{Pub\ Challenge}\ [\mathbf{end}(x)])$ in Gordon and Jeffrey's type system. The key *pk*$_B$ is public, but the $\mathbf{Pub}$ predicate does not hold for this type in Gordon and Jeffrey's type system [16].[7] The discrepancy comes from the fact that $\mathbf{Taint}(\mathbf{N_{Pub}}(\varphi, \emptyset))$ holds for arbitrary φ in our type system, but the corresponding condition $\mathbf{Taint}(\mathsf{Pub\ Challenge}\ \varphi)$ holds only for the case $\varphi = \emptyset$ in Gordon and Jeffrey's. This seems to be caused by the difference in the rules for typing check operations as discussed in the previous subsection. Because of the difference, allowing $\mathbf{Taint}(\mathsf{Pub\ Challenge}\ \varphi)$ to hold for arbitrary φ is unsound for Gordon and Jeffrey's type system.

C Proof of Lemma 1

Here we give a more detailed proof of Lemma 1. We define encodings of messages and processes. $[\![M]\!]_x$ translates a message M (that may not be well-typed) to a well-typed process that sends the value of M on channel x. $[\![P]\!]$ translates a process P to an equivalent, well-typed process. We assume below that renaming is applied as necessary

[7] Confirmed by email discussion with Gordon and Jeffrey.

to avoid the name clashing.

$$
\begin{array}{l}
[\![y]\!]_x = x!y \\
[\![(M_1, M_2)]\!]_x = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid y_1?z_1.y_2?z_2.x!(z_1, z_2)) \\
[\![\{M_1\}_{M_2}]\!]_x = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid y_1?z_1.y_2?z_2.x!\{z_1\}_{z_2}) \\
[\![\{\!|M_1|\!\}_{M_2}]\!]_x = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid y_1?z_1.y_2?z_2.x!\{\!|z_1|\!\}_{z_2})
\end{array}
$$

$$
\begin{array}{l}
[\![\mathbf{0}]\!] = \mathbf{0} \\
[\![M_1!M_2]\!] = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid y_1?z_1.y_2?z_2.z_1!z_2) \\
[\![M_1?x.P]\!] = (\nu y_1)([\![M_1]\!]_{y_1} \mid y_1?z_1.z_1?x.[\![P]\!]) \\
[\![*P]\!] = *[\![P]\!] \\
[\![(\nu x)P]\!] = (\nu x)[\![P]\!] \\
[\![(\nu_{sym} x)P]\!] = (\nu_{sym} x)[\![P]\!] \\
[\![(\nu_{asym} x_1, x_2)P]\!] = (\nu_{asym} x_1, x_2)[\![P]\!]
\end{array}
$$

$$
\begin{array}{l}
[\![\mathbf{check}\ M_1\ \mathbf{is}\ M_2.P]\!] = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid \\
\quad\quad y_1?z_1.y_2?z_2.\mathbf{check}\ z_1\ \mathbf{is}\ z_2.[\![P]\!]) \\
[\![\mathbf{split}\ M\ \mathbf{is}\ (x_1, x_2).P]\!] = \\
\quad (\nu y)([\![M]\!]_y \mid y?z.\mathbf{split}\ z\ \mathbf{is}\ (x_1, x_2).[\![P]\!]) \\
[\![\mathbf{match}\ M_1\ \mathbf{is}\ (M_2, x).P]\!] = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid \\
\quad\quad y_1?z_1.y_2?z_2.\mathbf{match}\ z_1\ \mathbf{is}\ (z_2, x).[\![P]\!]) \\
[\![\mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P]\!] = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid \\
\quad\quad y_1?z_1.y_2?z_2.\mathbf{decrypt}\ z_1\ \mathbf{is}\ \{x\}_{z_2}.[\![P]\!]) \\
[\![\mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P]\!] = \\
\quad (\nu y_1)(\nu y_2)([\![M_1]\!]_{y_1} \mid [\![M_2]\!]_{y_2} \mid \\
\quad\quad y_1?z_1.y_2?z_2.\mathbf{decrypt}\ z_1\ \mathbf{is}\ \{\!|x|\!\}_{z_2^{-1}}.[\![P]\!]) \\
[\![\mathbf{begin}\, M.P]\!] = \mathbf{begin}\, M.[\![P]\!] \\
[\![\mathbf{end}\, M]\!] = \mathbf{end}\, M
\end{array}
$$

By straightforward induction on the structures of M and P, we can prove:

$$y_1 : \mathbf{Un}, \ldots, y_n : \mathbf{Un}, x : \mathbf{Un}; \emptyset \vdash [\![M]\!]_x : \mathbf{Un}$$

and

$$z_1 : \mathbf{Un}, \ldots, z_m : \mathbf{Un}; \emptyset \vdash [\![P]\!]$$

where $\mathbf{FN}(M) = \{y_1, \ldots, y_n\}$ and $\mathbf{FN}(P) = \{z_1, \ldots, z_m\}$. It is also obvious that for any reduction sequence of $P \mid Q$, there is a corresponding $[\![P]\!] \mid Q$. Thus, the required result of the lemma holds for $O' = [\![O]\!]$.

D Proof of Lemma 2

D.1 Extended Processes and Typing

To prove Lemma 2, we extend the syntax of processes and the typing rules in order to express invariants preserved by reductions.

Extended Processes We extend the syntax of processes, in order to make it explicit what obligations and capabilities is carried by each name, and when they are attached to the name. We distinguish below between (bound) variables, ranged over by x, and (free) names, ranged over by n.

Definition D1 *The sets of* extended messages and processes *are given by:*

$$
\begin{array}{l}
M\textit{(ext. messages)} ::= \\
\quad v \mid \textbf{\textit{addC}}(M,\varphi_1,\varphi_2) \mid (M_1,M_2) \mid \{M_1\}_{M_2} \mid \{\!|M_1|\!\}_{M_2} \\
V\textit{(values)} ::= v \mid (V_1,V_2) \mid \{V_1\}_{V_2} \mid \{\!|V_1|\!\}_{V_2} \\
v ::= x \mid n^{(\varphi_1,\varphi_2)} \\
P\textit{(ext. processes)} ::= \\
\quad \mathbf{0} \mid M_1!M_2 \mid M?x.P \mid (P_1 \,|\, P_2) \mid *P \\
\mid (\nu x:\tau)P \mid (\nu_{sym} k:\tau)P \mid (\nu_{asym} k_1:\tau_1, k_2:\tau_2)P \\
\mid \mathbf{check}\ M_1\ \mathbf{is}\ M_2.P \\
\mid \mathbf{split}\ M\ \mathbf{is}\ (x,y).P \mid \mathbf{match}\ M_1\ \mathbf{is}\ (M_2,y).P \\
\mid \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P \mid \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P \\
\mid \mathbf{begin}\, V.P \mid \mathbf{end}\, V
\end{array}
$$

The typing rules for extended processes are shown in Figure 12 and 13. In Figure 12, $\leq_{\mathtt{ex}}$ is the least reflexive relation that satisfies the following rules:

$$\frac{\mathbf{Pub}(\tau) \qquad \mathbf{Taint}(\tau')}{\tau \leq_{\mathtt{ex}} \tau'} \qquad \text{(ExSubT-PubTaint)}$$

$$\frac{\varphi_1 \leq \varphi_1' \qquad \varphi_2 \geq \varphi_2'}{\mathbf{N}_\ell(\varphi_1,\varphi_2) \leq_{\mathtt{ex}} \mathbf{N}_\ell(\varphi_1',\varphi_2')} \qquad \text{(ExSubT-Name)}$$

We show properties of the subtyping relation below.

Lemma 3. *If* $\tau_1 \leq_{\mathtt{ex}} \tau_2$ *and* $\tau_2 \leq_{\mathtt{ex}} \tau_3$*, then* $\tau_1 \leq_{\mathtt{ex}} \tau_3$*.*

Proof. By a case analysis on the rules used for deriving $\tau_1 \leq_{\mathtt{ex}} \tau_2$ and $\tau_2 \leq_{\mathtt{ex}} \tau_3$. If one of the rules is reflexivity, the result follows immediately. There are four remaining cases.

– Case ExSubT-PubTaint-ExSubT-PubTaint: In this case, $\mathbf{Pub}(\tau_1)$ and $\mathbf{Taint}(\tau_3)$, from which the result follows by ExSubT-PubTaint.

– Case ExSubT-Name-ExSubT-Name: In this case, $\tau_i = \mathbf{N}_\ell(\varphi_i,\varphi_i')$ for $i \in \{1,2,3\}$ with $\varphi_1 \leq \varphi_2 \leq \varphi_3$ and $\varphi_1' \geq \varphi_2' \geq \varphi_3'$. Thus, the result follows by ExSubT-Name.

– Case ExSubT-PubTaint-ExSubT-Name: In this case, we have $\tau_i = \mathbf{N}_\ell(\varphi_i, \varphi'_i)$ for $i \in \{2, 3\}$ with $\mathbf{Pub}(\tau_1)$, $\mathbf{Taint}(\mathbf{N}_\ell(\varphi_2, \varphi'_2)$, $\varphi_2 \leq \varphi_3$ and $\varphi'_2 \geq \varphi_3$. If $\ell = \mathbf{Pub}$, then $\varphi'_2 = \emptyset$, which implies $\varphi_3 = \emptyset$. Thus, we have $\ell = \mathbf{Pub} \Rightarrow \varphi_3 = \emptyset$, which implies $\mathbf{Taint}(\tau_3)$. The required result is obtained by using ExSubT-PubTaint.

– Case ExSubT-Name-ExSubT-PubTaint: In this case, we have $\tau_i = \mathbf{N}_\ell(\varphi_i, \varphi'_i)$ for $i \in \{1, 2\}$ with $\mathbf{Taint}(\tau_3)$, $\mathbf{Pub}(\mathbf{N}_\ell(\varphi_2, \varphi'_2)$, $\varphi_1 \leq \varphi_2$ and $\varphi'_1 \geq \varphi_2$. By the condition $\mathbf{Pub}(\mathbf{N}_\ell(\varphi_2, \varphi'_2))$, we have $\ell = \mathbf{Pub}$ and $\varphi_2 = \emptyset$, which implies $\varphi_1 = \emptyset$. Thus, we have $\mathbf{Pub}(\tau_1)$. The required result follows by ExSubT-PubTaint.

The following lemma guarantees that the subsumption rule (ExT-Sub) only increases obligations, and decreases capabilities of a name type, unless the qualification of the name type is changed from $\mathbf{Pub}$ to $\mathbf{Pr}$.

Lemma 4. *If* $\mathbf{N}_\ell(\varphi_1, \varphi_2) \leq_{\mathsf{ex}} \mathbf{N}_{\ell'}(\varphi'_1, \varphi'_2)$*, then either* $\ell = \mathbf{Pub} \wedge \ell' = \mathbf{Pr} \wedge \varphi_1 = \emptyset$*, or* $\ell = \ell' \wedge \varphi_1 \leq \varphi'_1 \wedge \varphi_2 \geq \varphi'_2$*.*

Proof. In the case where $\mathbf{N}_\ell(\varphi_1, \varphi_2) \leq_{\mathsf{ex}} \mathbf{N}_{\ell'}(\varphi'_1, \varphi'_2)$ was derived using rule ExSubT-Name we immediately have that the second set of conditions are satisfied. If instead rule ExSubT-PubTaint was used we first note that in this case $\ell = \mathbf{Pub}$ must hold. Then, if $\ell' = \mathbf{Pub}$ (respectively $\mathbf{Pr}$) we have that the second (respectively first) set of conditions are satisfied.

Extended Operational Semantics

Definition D2 *The set of* message reduction contexts *for messages, ranged over by* C_m*, is given by:*

$$\begin{aligned} C_m ::= {} & [\,] \mid \textbf{\textit{addC}}(C_m, \varphi_1, \varphi_2) \mid (C_m, M) \mid (V, C_m) \mid \{C_m\}_M \\ & \mid \{V\}_{C_m} \mid \{\!|C_m|\!\}_M \mid \{\!|V|\!\}_{C_m} \end{aligned}$$

The message reduction relation is defined by:

$$\begin{aligned} & (\varphi + \varphi'_1 + \varphi'_2, C_m[\textbf{\textit{addC}}(n^{(\varphi_1,\varphi_2)}, \varphi'_1, \varphi'_2)]) \\ & \qquad \longrightarrow_{\mathsf{ex}} (\varphi, C_m[n^{(\varphi_1 - \varphi'_1, \varphi_2 + \varphi'_2)}]) \end{aligned}$$

$$\begin{aligned} & (\varphi, C_m[\textbf{\textit{addC}}(n^{(\varphi_1,\varphi_2)}, \varphi'_1, \varphi'_2)]) \longrightarrow_{\mathsf{ex}} \mathbf{Error} \\ & \qquad\qquad\qquad\qquad \textit{(if } \varphi'_1 + \varphi'_2 \not\leq \varphi\textit{)} \end{aligned}$$

The extended reduction relation $\langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle \longrightarrow_{\mathsf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi' \rangle$ is defined by the rules in Figure 11. Here, Ψ is a multiset of extended processes, E is a set of messages (that represent the begin-events that have occured but have not been matched by corresponding end-events), Γ is a set of names that have been created (with type assumptions), $\mathcal{K}$ is a set of pairs of decryption and encryption keys, and φ is a capability.

In the figure, C ranges over the set of reduction contexts (for messages), given by:

$$\begin{aligned}
C ::= {} & M_1!M_2 \mid v!C_m \mid M?x.P \mid (P_1 \mid P_2) \\
& \mid \textbf{check } C_m \textbf{ is } M.P \mid \textbf{check } V \textbf{ is } C_m.P \\
& \mid \textbf{split } C_m \textbf{ is } (x, y).P \\
& \mid \textbf{match } C_m \textbf{ is } (M, y).P \mid \textbf{match } V \textbf{ is } (C_m, y).P \\
& \mid \textbf{decrypt } C_m \textbf{ is } \{x\}_{M_2}.P \mid \textbf{decrypt } V \textbf{ is } \{x\}_{C_m}.P \\
& \mid \textbf{decrypt } C_m \textbf{ is } \{\!|x|\!\}_{M_2{}^{-1}}.P \\
& \mid \textbf{decrypt } V \textbf{ is } \{\!|x|\!\}_{C_m{}^{-1}}.P
\end{aligned}$$

D.2 Proof

For an extended process P, we write $\mathbf{Erase}(P)$ for the process obtained by removing type annotations and "addC".

Lemma 5. *If $\Gamma; \varphi \vdash P$, then there exists P' such that $\Gamma; \varphi \vdash_{\mathbf{ex}} P'$ and $\mathbf{Erase}(P') = P$.*

Proof. Easy induction on the derivation of $\Gamma; \varphi \vdash P$.

Lemma 6. *If $\langle P, \emptyset, \emptyset, \emptyset, \emptyset \rangle \not\longrightarrow^*_{\mathbf{ex}} \mathbf{Error}$, then $\mathbf{Erase}(P)$ is safe.*

Proof. We show contraposition. Suppose $\mathbf{Erase}(P)$ is not safe, i.e., $\langle \mathbf{Erase}(P), \emptyset, \emptyset, \emptyset \rangle \longrightarrow^*$ $\mathbf{Error}$. Then $\langle P, \emptyset, \emptyset, \emptyset, \emptyset \rangle \longrightarrow^*_{\mathbf{ex}} \mathbf{Error}$ follows from the facts: (i) if $\langle \mathbf{Erase}(P), E, dom(\Gamma), \mathcal{K} \rangle \longrightarrow \langle Q, E', N', \mathcal{K}' \rangle$, then either $\langle P, E, \Gamma, \mathcal{K}, \varphi \rangle \longrightarrow_{\mathbf{ex}} \langle P', E', \Gamma', \mathcal{K}', \varphi' \rangle$ with $\mathbf{Erase}(P') = P$ and $dom(\Gamma') = N'$ or $\langle P, E, \Gamma, \mathcal{K}, \varphi \rangle \longrightarrow_{\mathbf{ex}} \mathbf{Error}$; and (ii) if $\langle \mathbf{Erase}(P), E, dom(\Gamma), \mathcal{K} \rangle \longrightarrow \mathbf{Error}$, then $\langle P, E, \Gamma, \mathcal{K}, \varphi \rangle \longrightarrow^*_{\mathbf{ex}} \mathbf{Error}$. (These facts follow by an easy case analysis on the rule used for deriving $\langle \mathbf{Erase}(P), E, dom(\Gamma), \mathcal{K} \rangle \longrightarrow \langle Q, E', N', \mathcal{K}' \rangle$ or $\langle \mathbf{Erase}(P), E, dom(\Gamma), \mathcal{K} \rangle \longrightarrow \mathbf{Error}$.)

Lemma 7. *If $\emptyset; \emptyset \vdash_{\mathbf{ex}} P$, then $\langle P, \emptyset, \emptyset, \emptyset, \emptyset \rangle \not\longrightarrow^*_{\mathbf{ex}} \mathbf{Error}$.*

To show Lemma 7, we define a typing rule for run-time configurations (of the form $\langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$) and show (i) the typing is preserved (Lemma 11) and (ii) a well-typed configuration is not immediately reduced to $\mathbf{Error}$ (Lemma 8).

In order to express a necessary invariant, we introduce a reduction relation $(\varphi, N) \Longrightarrow_{\Psi} (\varphi', N')$, which is used to collect all the capabilities, including those attached to names. Here, the second component N is a set of names, used to keep track of the names that have been checked.

$$\frac{n^{(\varphi_3, \varphi_4)} \text{ occurs in } \Psi}{\begin{aligned}&(\varphi_1 + \{\mathbf{chk}_\ell(n, \varphi_2) \mapsto 1\}, N) \\ &\quad \Longrightarrow_{\Psi} (\varphi_1 + (\varphi_2 - \varphi_3) + \varphi_4, N \cup \{n\})\end{aligned}}$$

We write **ConsistentCap**(E, φ, Ψ) if, whenever $(\varphi, \emptyset) \Longrightarrow^*_{\Psi} (\varphi', N)$, the following conditions hold: (i) $E(V) \geq \varphi'(\mathbf{end}(V))$ for every V, and (ii) for every n, if $\varphi'(\mathbf{chk}_\ell(n, \varphi_2)) \geq 1$, then $n \notin N$.

$$\frac{(\varphi, M) \longrightarrow_{\mathbf{ex}} (\varphi', M')}{\langle \Psi \uplus \{C[M]\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{C[M']\}, E, \Gamma, \mathcal{K}, \varphi'\rangle} \quad \text{(ExR-M)}$$

$$\frac{(\varphi, M) \longrightarrow_{\mathbf{ex}} \mathbf{Error}}{\langle \Psi \uplus \{C[M]\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \mathbf{Error}} \quad \text{(ExR-M-Er)}$$

$$\langle \Psi \uplus \{n^{(\_,\_)}?y.P, n^{(\_,\_)}!V\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{[V/y]P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \quad \text{(ExR-Com)}$$

$$\langle \Psi \uplus \{P \mid Q\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{P, Q\}, E, \Gamma, \mathcal{K}, \varphi\rangle \quad \text{(ExR-Par)}$$

$$\langle \Psi \uplus \{*P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{*P, P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \quad \text{(ExR-Rep)}$$

$$\langle \Psi \uplus \{(\nu x : \mathbf{N}_\ell(\varphi_1, \emptyset))P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}}$$
$$\langle \Psi \uplus \{[n^{(\varphi_1,\emptyset)}/x]P\}, E, \Gamma \cup \{n : \mathbf{N}_\ell(\varphi_1, \emptyset)\}, \mathcal{K}, \varphi + \{\mathbf{chk}_\ell(n, \varphi_1)\}\rangle \quad (n \notin dom(\Gamma))$$
(ExR-NewN)

$$\langle \Psi \uplus \{(\nu_{sym} x : \tau)P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}}$$
$$\langle \Psi \uplus \{[k/x]P\}, E, \Gamma \cup \{k : \tau\}, \mathcal{K}, \varphi\rangle \ (k \notin dom(\Gamma)) \quad \text{(ExR-NewSk)}$$

$$\langle \Psi \uplus \{(\nu_{asym} x : \tau_1, y : \tau_2)P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}}$$
$$\langle \Psi \uplus \{[k_1/x, k_2/y]P\}, E, \Gamma \cup \{k_1 : \tau_1, k_2 : \tau_2\}, \mathcal{K} \cup \{(k_1, k_2)\}, \varphi\rangle \ (k_1, k_2 \notin dom(\Gamma))$$
(ExR-NewAk)

$$\langle \Psi \uplus \{\mathbf{check}\, n^{(\_,\_)} \ \mathbf{is}\ n^{(\emptyset,\varphi_1)}.P\}, E, \Gamma, \mathcal{K}, \varphi + \{\mathbf{chk}_\ell(n, \varphi_2)\}\rangle \longrightarrow_{\mathbf{ex}}$$
$$\langle \Psi \uplus \{P\}, E, \Gamma, \mathcal{K}, \varphi + \varphi_1 + \varphi_2\rangle \quad \text{(ExR-Chk)}$$

$$\frac{(\varphi_0 \neq \emptyset) \vee \neg\exists \varphi_2, \ell.(\mathbf{chk}_\ell(n, \varphi_2) \in \varphi)}{\langle \Psi \uplus \{\mathbf{check}\, n^{(\_,\_)} \ \mathbf{is}\ n^{(\varphi_0,\varphi_1)}.P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \mathbf{Error}} \quad \text{(ExR-Chk-Er)}$$

$$\langle \Psi \uplus \{\mathbf{split}\ (V, W)\ \mathbf{is}\ (x, y).P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{[V/x, W/y]P\}, E, \Gamma, \mathcal{K}, \varphi\rangle$$
(ExR-Splt)

$$\langle \Psi \uplus \{\mathbf{match}\ (V, W)\ \mathbf{is}\ (V, z).P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{[W/z]P\}, E, \Gamma, \mathcal{K}, \varphi\rangle$$
(ExR-Mtch)

$$\langle \Psi \uplus \{\mathbf{decrypt}\ \{V\}_k\ \mathbf{is}\ \{x\}_k.P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{[V/x]P\}, E, \Gamma, \mathcal{K}, \varphi\rangle$$
(ExR-SDec)

$$\frac{(k_1, k_2) \in \mathcal{K}}{\langle \Psi \uplus \{\mathbf{decrypt}\ \{\!|V|\!\}_{k_1}\ \mathbf{is}\ \{\!|x|\!\}_{k_2^{-1}}.P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{[V/x]P\}, E, \Gamma, \mathcal{K}, \varphi\rangle}$$
(ExR-ADec)

$$\langle \Psi \uplus \{\mathbf{begin}\, V.P\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi \uplus \{P\}, E \uplus \{V\}, \Gamma, \mathcal{K}, \varphi + \{\mathbf{end}(V)\}\rangle$$
(ExR-Beg)

$$\langle \Psi \uplus \{\mathbf{end}\, V\}, E \uplus \{V\}, \Gamma, \mathcal{K}, \varphi + \{\mathbf{end}(V)\}\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle \quad \text{(ExR-End)}$$

$$\frac{(V \notin E) \vee (\varphi(\mathbf{end}(V)) < 1)}{\langle \Psi \uplus \{\mathbf{end}\, V\}, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \mathbf{Error}} \quad \text{(ExR-End-Er)}$$

Fig. 11. Extended Operational Semantics

$$\Gamma, x:\tau;\varphi \vdash_{\mathbf{ex}} x:\tau \qquad \text{(ExT-Var)}$$

$$\Gamma, n:\mathbf{N}_\ell(\_,\_);\varphi \vdash_{\mathbf{ex}} n^{(\varphi_1,\varphi_2)}:\mathbf{N}_\ell(\varphi_1,\varphi_2) \qquad \text{(ExT-Name)}$$

$$\frac{\Gamma;\varphi \vdash_{\mathbf{ex}} V:\mathbf{N}_\ell(\varphi_1,\varphi_2)}{\Gamma;\varphi+\varphi_1'+\varphi_2' \vdash_{\mathbf{ex}} \mathbf{addC}(V,\varphi_1',\varphi_2'):\mathbf{N}_\ell(\varphi_1-\varphi_1',\varphi_2+\varphi_2')} \qquad \text{(ExT-AddC)}$$

$$\frac{\Gamma;\varphi_1 \vdash_{\mathbf{ex}} M_1:\tau_1 \qquad \Gamma;\varphi_2 \vdash_{\mathbf{ex}} M_2:[M_1/0]\tau_2}{\Gamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} (M_1,M_2):\tau_1\times\tau_2} \qquad \text{(ExT-Pair)}$$

$$\frac{\Gamma;\varphi_1 \vdash_{\mathbf{ex}} M_1:\tau_1 \qquad \Gamma;\varphi_2 \vdash_{\mathbf{ex}} M_2:\mathbf{SKey}(\tau_1)}{\Gamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} \{M_1\}_{M_2}:\mathbf{N}_\ell(\emptyset,\emptyset)} \qquad \text{(ExT-SEnc)}$$

$$\frac{\Gamma;\varphi_1 \vdash_{\mathbf{ex}} M_1:\tau \qquad \Gamma;\varphi_2 \vdash_{\mathbf{ex}} M_2:\mathbf{EKey}(\tau)}{\Gamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} \{\!|M_1|\!\}_{M_2}:\mathbf{N}_\ell(\emptyset,\emptyset)} \qquad \text{(ExT-AEnc)}$$

$$\frac{\Gamma;\varphi \vdash_{\mathbf{ex}} M:\tau' \qquad \tau' \leq_{\mathrm{ex}} \tau}{\Gamma;\varphi \vdash_{\mathbf{ex}} M:\tau} \qquad \text{(ExT-Sub)}$$

Fig. 12. Typing Rules for Extended Messages

The condition (i) ensures that the end capabilities estimated by the type system (i.e. φ') is at most those that are actually present (E). The condition (ii) ensures that there is always at most one check capability for each name.

The typing rule for run-time configurations is given as follows:

$$\frac{\begin{array}{c}\Gamma;\varphi \vdash_{\mathbf{ex}} P_1 \mid \cdots \mid P_m \\ \textbf{ConsistentCap}(E,\varphi,\{P_1,\ldots,P_m\}) \\ \forall (k_1,k_2)\in\mathcal{K}.\exists\tau.(\Gamma(k_1)=\mathbf{EKey}(\tau)\wedge\Gamma(k_2)=\mathbf{DKey}(\tau)) \\ \forall n,\ell,\ell'.(\Gamma(n)=\mathbf{N}_\ell(\_,\_)\wedge \\ (\text{“}\mathbf{chk}_{\ell'}(n,\_)\text{ occurs in some } P_i \text{ or } \varphi\text{”})\Rightarrow \ell=\ell')\end{array}}{\vdash_{\mathbf{ex}} \langle\{P_1,\ldots,P_m\},E,\Gamma,\mathcal{K},\varphi\rangle}$$

Lemma 7 follows from Lemmas 8 and 11 below.

Lemma 8 (lack of immediate error).
If $\vdash_{\mathbf{ex}} \langle\Psi,E,\Gamma,\mathcal{K},\varphi\rangle$*, then* $\langle\Psi,E,\Gamma,\mathcal{K},\varphi\rangle \not\longrightarrow_{\mathbf{ex}} \mathbf{Error}$.

Proof. Suppose $\vdash_{\mathbf{ex}} \langle\Psi,E,\Gamma,\mathcal{K},\varphi\rangle$ holds. There are three rules that may yield $\mathbf{Error}$: ExR-M-Er, ExR-Chk-Er, and ExR-End-Er. We show below that none of those rules is applicable.

- Case ExR-M-Er: In this case, $\Psi=\Psi'\uplus\{C[M]\}$ with $(\varphi,M)\longrightarrow_{\mathbf{ex}}\mathbf{Error}$. By the typing rules, it must be the case that $\Gamma;\varphi'\vdash_{\mathbf{ex}} M:\tau$ and $\varphi'\subseteq\varphi$ for some φ' and τ. By the typing rules and reduction rules for messages, $(\varphi,M)\longrightarrow_{\mathbf{ex}}\mathbf{Error}$ cannot hold.
- Case ExR-Chk-Er: In this case, $\Psi=\Psi'\uplus\{\mathbf{check}\ n^{(\_,\_)}\ \mathbf{is}\ n^{(\varphi_0,\varphi_1)}.P\}$. By the assumption $\vdash_{\mathbf{ex}} \langle\Psi,E,\Gamma,\mathcal{K},\varphi\rangle$, there must exist φ' such that $\varphi'\leq\varphi$ and

$$\Gamma;\varphi'\vdash_{\mathbf{ex}} \mathbf{check}\ n^{(\_,\_)}\ \mathbf{is}\ n^{(\varphi_0,\varphi_1)}.P.$$

$$\varGamma;\emptyset \vdash_{\mathbf{ex}} \mathbf{0}$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M_1 : \mathbf{N}_\ell(\emptyset,\emptyset) \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} M_2 : \tau \qquad \mathbf{Pub}(\tau)}{\varGamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} M_1!M_2} \quad (\text{ExT-Out})$$

$$\frac{\varGamma_1;\varphi_1 \vdash_{\mathbf{ex}} M : \mathbf{N}_\ell(\emptyset,\emptyset) \qquad \varGamma_2, x:\tau;\varphi_2 \vdash_{\mathbf{ex}} P \qquad \mathbf{Taint}(\tau)}{\varGamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} M?x.P} \quad (\text{ExT-In})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} P_1 \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} P_2}{\varGamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} P_1 \mid P_2} \quad (\text{ExT-Par})$$

$$\frac{\varGamma;\emptyset \vdash_{\mathbf{ex}} P}{\varGamma;\emptyset \vdash_{\mathbf{ex}} *P} \quad (\text{ExT-Rep})$$

$$\frac{\varGamma, x:\mathbf{N}_\ell(\varphi_1,\emptyset);\varphi+\{\mathbf{chk}_\ell(x,\varphi_1)\mapsto 1\} \vdash_{\mathbf{ex}} P}{\varGamma;\varphi \vdash_{\mathbf{ex}} (\nu x:\mathbf{N}_\ell(\varphi_1,\emptyset))P} \quad (\text{ExT-NewN})$$

$$\frac{\varGamma, x:\mathbf{SKey}(\tau);\varphi \vdash_{\mathbf{ex}} P}{\varGamma;\varphi \vdash_{\mathbf{ex}} (\nu_{sym} x:\mathbf{SKey}(\tau))P} \quad (\text{ExT-NewSk})$$

$$\frac{\varGamma, k_1:\mathbf{EKey}(\tau), k_2:\mathbf{DKey}(\tau);\varphi \vdash_{\mathbf{ex}} P}{\varGamma;\varphi \vdash_{\mathbf{ex}} (\nu_{asym} k_1:\mathbf{EKey}(\tau), k_2:\mathbf{DKey}(\tau))P} \quad (\text{ExT-NewAk})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M_1 : \mathbf{N}_\ell(\_,\_) \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} M_2 : \mathbf{SKey}(\tau) \qquad \varGamma, x:\tau;\varphi_3 \vdash_{\mathbf{ex}} P}{\varGamma;\varphi_1+\varphi_2+\varphi_3 \vdash_{\mathbf{ex}} \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{x\}_{M_2}.P} \quad (\text{ExT-SDec})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M_1 : \mathbf{N}_\ell(\_,\_) \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} M_2 : \mathbf{DKey}(\tau) \qquad \varGamma, x:\tau;\varphi_3 \vdash_{\mathbf{ex}} P}{\varGamma;\varphi_1+\varphi_2+\varphi_3 \vdash_{\mathbf{ex}} \mathbf{decrypt}\ M_1\ \mathbf{is}\ \{\!|x|\!\}_{M_2^{-1}}.P} \quad (\text{ExT-ADec})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M_1 : \mathbf{N}_\ell(\_,\_) \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} M_2 : \mathbf{N}_\ell(\emptyset,\varphi_5) \qquad \varGamma;\varphi_3+\varphi_4+\varphi_5 \vdash_{\mathbf{ex}} P}{\varGamma;\varphi_1+\varphi_2+\varphi_3+\{\mathbf{chk}_\ell(M_1,\varphi_4)\} \vdash_{\mathbf{ex}} \mathbf{check}\ M_1\ \mathbf{is}\ M_2.P} \quad (\text{ExT-Chk})$$

$$\frac{\varGamma;\varphi+\{\mathbf{end}(V)\mapsto 1\} \vdash_{\mathbf{ex}} P}{\varGamma;\varphi \vdash_{\mathbf{ex}} \mathbf{begin}\,V.P} \quad (\text{ExT-Beg})$$

$$\frac{}{\varGamma;\varphi+\{\mathbf{end}(V)\mapsto 1\} \vdash_{\mathbf{ex}} \mathbf{end}\,V} \quad (\text{ExT-End})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M : \tau_1\times\tau_2 \qquad \varGamma, y:\tau_1, z:[y/0]\tau_2;\varphi_2 \vdash_{\mathbf{ex}} P}{\varGamma;\varphi_1+\varphi_2 \vdash_{\mathbf{ex}} \mathbf{split}\ M\ \mathbf{is}\ (y,z).P} \quad (\text{ExT-Splt})$$

$$\frac{\varGamma;\varphi_1 \vdash_{\mathbf{ex}} M_1 : \tau_1\times\tau_2 \qquad \varGamma;\varphi_2 \vdash_{\mathbf{ex}} M_2 : \tau_1 \qquad \varGamma, z:[M_2/0]\tau_2;\varphi_3 \vdash_{\mathbf{ex}} P}{\varGamma;\varphi_1+\varphi_2+\varphi_3 \vdash_{\mathbf{ex}} \mathbf{match}\ M_1\ \mathbf{is}\ (M_2,z).P} \quad (\text{ExT-Mtch})$$

$$\frac{\varGamma;\varphi' \vdash_{\mathbf{ex}} P \qquad \varphi' \le \varphi}{\varGamma;\varphi \vdash_{\mathbf{ex}} P} \quad (\text{ExT-WeakCap})$$

Fig. 13. Typing Rules for Extended Processes

By the typing rules, we have:

$$\begin{array}{l}\Gamma; \varphi_2 \vdash_{\mathbf{ex}} n^{(\_,\_)} : \mathbf{N}_\ell(\_,\_) \\ \Gamma; \varphi_3 \vdash_{\mathbf{ex}} n^{(\varphi_0,\varphi_1)} : \mathbf{N}_\ell(\emptyset, \varphi_5) \\ \Gamma; \varphi_4 + \varphi_5 + \varphi_6 \vdash_{\mathbf{ex}} P \\ \varphi' \geq \varphi_2 + \varphi_3 + \varphi_4 + \{\mathbf{chk}_\ell(n, \varphi_6)\}\end{array}$$

By the second condition, we have $\varphi_0 = \emptyset$. (Note that the judgment must have been derived from EXT-NAME, followed by a possible application of EXT-SUB. EXT-NAME assigns the type $\mathbf{N}_{\ell'}(\varphi_0, \varphi_1)$, and by Lemma 3, we must have $\mathbf{N}_{\ell'}(\varphi_0, \varphi_1) \leq_{\mathrm{ex}} \mathbf{N}_\ell(\emptyset, \varphi_5)$. By Lemma 4, we have $\varphi_0 = \emptyset$.) Thus, the premise of EXR-CHK-ER does not hold.

– Case EXR-END-ER: In this case, $\Psi = \Psi' \uplus \{\mathbf{end}\, V\}$ with $(V \notin E) \vee (\varphi(\mathbf{end}(V)) < 1)$. If $V \notin E$, then by the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle$ and the second condition on the configuration typing, we have $(\varphi(\mathbf{end}(V)) < 1)$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle$, however, we also have $\Gamma; \varphi' \vdash_{\mathbf{ex}} \mathbf{end}\, V$ for some $\varphi' \leq \varphi$. By the typing rule for $\mathbf{end}\, V$, it must be the case that $(\varphi'(\mathbf{end}(V)) \geq 1)$, hence a contradiction.

Lemma 9. *If* $\Gamma; \varphi \vdash_{\mathbf{ex}} V : \tau$*, then* $\Gamma; \emptyset \vdash_{\mathbf{ex}} V : \tau$*.*

Proof. Straightforward induction on the derivation of $\Gamma; \varphi \vdash_{\mathbf{ex}} V : \tau$. (Note that by the syntax of values, V does not contain “addC”.)

Lemma 10 (substitution). *If* $\Gamma_1; \emptyset \vdash_{\mathbf{ex}} V : \tau$ *and* $\Gamma_1, x : \tau, \Gamma_2; \varphi \vdash_{\mathbf{ex}} P$*, then* $\Gamma_1, [V/x]\Gamma_2; [V/x]\varphi \vdash_{\mathbf{ex}} [V/x]P$*.*

Proof. A derivation of $\Gamma_1, [V/x]\Gamma_2; [V/x]\varphi \vdash_{\mathbf{ex}} [V/x]P$ is obtained from $\Gamma_1, x{:}\tau, \Gamma_2; \varphi \vdash_{\mathbf{ex}} P$ by replacing each leaf of the form $\Gamma_1, x : \tau, \Gamma_2'; \varphi' \vdash_{\mathbf{ex}} x : \tau$ (where $\Gamma_2' \supseteq \Gamma_2$) with $\Gamma_1, [V/x]\Gamma_2; [V/x]\varphi' \vdash_{\mathbf{ex}} V : \tau$ (which is obtained by weakening and EXT-WEAKCAP).

Lemma 11 (type preservation). *If* $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle$ *and* $\langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi'\rangle$*, then* $\vdash_{\mathbf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi'\rangle$*.*

Proof. Suppose $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle$ and $\langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi'\rangle$. We show $\vdash_{\mathbf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi'\rangle$ by case analysis on the rule used for deriving $\langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle \longrightarrow_{\mathbf{ex}} \langle \Psi', E', \Gamma', \mathcal{K}', \varphi'\rangle$. By abuse of notation, we often write $\Gamma; \varphi \vdash \{P_1, \ldots, P_k\}$ for $\Gamma; \varphi \vdash P_1 \mid \cdots \mid P_k$ below.

– Case EXR-M: In this case, $\Psi = \Psi_1 \uplus \{C[M]\}$ and $\Psi' = \Psi_1 \uplus \{C[M']\}$ with $M = C_m[\mathbf{addC}(n^{(\varphi_1,\varphi_2)}, \varphi_1', \varphi_2')]$, $M' = C_m[n^{(\varphi_1 - \varphi_1', \varphi_2 + \varphi_2')}]$, $\varphi = \varphi' + \varphi_1' + \varphi_2'$. We also have $E' = E$, $\Gamma' = \Gamma$, and $\mathcal{K}' = \mathcal{K}$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi\rangle$, $\Gamma; \varphi \vdash \Psi_1 \uplus \{C[C_m[\mathbf{addC}(n^{(\varphi_1,\varphi_2)}, \varphi_1', \varphi_2')]]\}$ holds, which must have been derived from $\Gamma; \varphi_1' + \varphi_2' \vdash_{\mathbf{ex}} \mathbf{addC}(n^{(\varphi_1,\varphi_2)}, \varphi_1', \varphi_2') : \mathbf{N}_{\ell'}(\varphi_1'' - \varphi_1', \varphi_2'' + \varphi_2')$, where $\Gamma(n) = \mathbf{N}_\ell(\_,\_)$ and $\mathbf{N}_\ell(\varphi_1, \varphi_2) \leq_{\mathrm{ex}} \mathbf{N}_{\ell'}(\varphi_1'', \varphi_2'')$. By Lemma 4, we have either $\ell = \mathbf{Pub} \wedge \ell' = \mathbf{Taint} \wedge \varphi_1 = \emptyset$, or $\ell = \ell' \wedge \varphi_1 \leq \varphi_1'' \wedge \varphi_2 \geq \varphi_2''$. In both cases, we have $\mathbf{N}_\ell(\varphi_1 - \varphi_1', \varphi_2 + \varphi_2') \leq_{\mathrm{ex}} \mathbf{N}_{\ell'}(\varphi_1'' - \varphi_1', \varphi_2'' + \varphi_2')$, which implies $\Gamma; \emptyset \vdash_{\mathbf{ex}} n^{(\varphi_1 - \varphi_1', \varphi_2 + \varphi_2')} : \mathbf{N}_{\ell'}(\varphi_1'' - \varphi_1', \varphi_2'' + \varphi_2')$. Thus, we have

$$\Gamma; \varphi' \vdash \Psi_1 \uplus \{C[C_m[n^{(\varphi_1 - \varphi_1', \varphi_2 + \varphi_2')}]]\}.$$

It remains to check **ConsistentCap**(E, φ', Ψ'). To check this, it suffices to observe that whenever $(\varphi', \emptyset) \Longrightarrow^*_{\Psi'} (\varphi'_3, \varphi'_4)$, we can construct a corresponding sequence $(\varphi, \emptyset) \Longrightarrow^*_{\Psi} (\varphi_3, \varphi_4)$ such that $\varphi'_3 + \varphi'_4 \leq \varphi_3 + \varphi_4$. (The only reduction step $(\varphi', \emptyset) \Longrightarrow^*_{\Psi'} (\varphi'_3, \varphi'_4)$ introduces more capabilities is a reduction on $\mathbf{chk}_\ell(n, \_)$, but that can happen at most once, and the difference is at most $\varphi'_1 + \varphi'_2$.)

– Case ExR-Com: In this case, $\Psi = \Psi_1 \uplus \{n^{(\_,\_)}?y.P, n^{(\_,\_)}!V\}$ and $\Psi = \Psi_1 \uplus \{[V/y]P\}$, with $E' = E$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, and $\varphi' = \varphi$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l}
\Gamma; \varphi_1 \vdash_{\mathbf{ex}} \Psi_1 \\
\Gamma; \varphi_2 \vdash_{\mathbf{ex}} n^{(\_,\_)} : \mathbf{N}_\ell(\emptyset, \emptyset) \\
\Gamma; \varphi_3 \vdash_{\mathbf{ex}} V : \tau \\
\mathbf{Pub}(\tau) \\
\Gamma; \varphi_4 \vdash_{\mathbf{ex}} n^{(\_,\_)} : \mathbf{N}_{\ell'}(\emptyset, \emptyset) \\
\Gamma, y : \tau'; \varphi_5 \vdash_{\mathbf{ex}} P \\
\mathbf{Taint}(\tau') \\
\varphi \geq \varphi_1 + \varphi_2 + \varphi_3 + \varphi_4 + \varphi_5
\end{array}$$

By the conditions $\Gamma; \varphi_3 \vdash_{\mathbf{ex}} V : \tau$, $\mathbf{Pub}(\tau)$, and $\mathbf{Taint}(\tau')$, we have $\Gamma; \varphi_3 \vdash_{\mathbf{ex}} V : \tau'$. By Lemma 9, $\Gamma; \emptyset \vdash_{\mathbf{ex}} V : \tau'$ holds. Thus, by using the substitution lemma (Lemma 10), we obtain $\Gamma; \varphi_5 \vdash_{\mathbf{ex}} [V/y]P$. By using ExT-Par and ExT-WeakCap, we obtain $\Gamma; \varphi \vdash_{\mathbf{ex}} \Psi'$ as required. **ConsistentCap**(E, φ, Ψ') follows immediately from **ConsistentCap**(E, φ, Ψ).

– Case ExR-Par, ExR-Rep: Trivial.

– Case ExR-NewN: In this case, $\Psi = \Psi_1 \uplus \{(\nu x : \mathbf{N}_\ell(\varphi_1, \emptyset))P\}$ and $\Psi' = \Psi_1 \uplus \{[n^{(\varphi_1, \emptyset)}/x]P\}$, with $E' = E$, $\Gamma' = (\Gamma, n : \mathbf{N}_\ell(\varphi_1, \emptyset))$, $\mathcal{K}' = \mathcal{K}$, and $\varphi' = \varphi + \{\mathbf{chk}_\ell(n, \varphi_1) \mapsto 1\}$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$ and rule ExT-NewN we have: $\Gamma; \varphi_2 \vdash_{\mathbf{ex}} \Psi_1$ and $\Gamma, x : \mathbf{N}_\ell(\varphi_1, \emptyset); \varphi_3 + \{\mathbf{chk}_\ell(x, \varphi_1) \mapsto 1\} \vdash_{\mathbf{ex}} P$, with $\varphi \geq \varphi_2 + \varphi_3$. By the substitution lemma (Lemma 10), we have

$$\Gamma'; \varphi_3 + \{\mathbf{chk}_\ell(n, \varphi_1) \mapsto 1\} \vdash_{\mathbf{ex}} [n^{(\varphi_1, \emptyset)}/x]P.$$

Thus, by using ExT-Par and ExT-WeakCap, we obtain $\Gamma'; \varphi' \vdash_{\mathbf{ex}} \Psi'$ as required. **ConsistentCap**(E', φ', Ψ') follows immediately from **ConsistentCap**(E, φ, Ψ).

– Case ExR-NewSk: Similar to the case for ExR-New.

– Case ExR-NewAk: Similar to the case for ExR-New.

– Case ExR-Chk: In this case, $\Psi = \Psi_1 \uplus \{\mathbf{check}\ n^{(\_,\_)}\ \mathbf{is}\ n^{(\emptyset, \varphi_1)}.P\}$ and $\Psi' = \Psi_1 \uplus \{P\}$, with $E' = E$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, $\varphi = \varphi_0 + \{\mathbf{chk}_\ell(n, \varphi_2)\}$ and $\varphi' = \varphi_0 + \varphi_1 + \varphi_2$.
By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l}
\Gamma; \varphi_3 \vdash_{\mathbf{ex}} \Psi_1 \\
\Gamma; \varphi_4 \vdash_{\mathbf{ex}} n^{(\_,\_)} : \mathbf{N}_\ell(\_, \_) \\
\Gamma; \varphi_5 \vdash_{\mathbf{ex}} n^{(\varphi_7, \varphi_8)} : \mathbf{N}_\ell(\emptyset, \varphi_1) \\
\quad \text{(with } \mathbf{N}_\ell(\varphi_7, \varphi_8) \leq_{\mathbf{ex}} \mathbf{N}_\ell(\emptyset, \varphi_1)) \\
\Gamma; \varphi_6 + \varphi_1 + \varphi_2 \vdash_{\mathbf{ex}} P \\
\varphi_0 \geq \varphi_3 + \varphi_4 + \varphi_5 + \varphi_6
\end{array}$$

Therefore, we have $\Gamma; \varphi_3 + \varphi_6 + \varphi_1 + \varphi_2 \vdash_{\mathbf{ex}} \Psi'$. By EXT-WEAKCAP, we obtain $\Gamma; \varphi' \vdash_{\mathbf{ex}} \Psi'$. It remains to check **ConsistentCap**(E, φ', Ψ').
Next, we show that $\varphi_7 = \emptyset$ and $\varphi_8 \geq \varphi_1$. By the condition $\mathbf{N}_\ell(\varphi_7, \varphi_8) \leq_{\mathtt{ex}} \mathbf{N}_\ell(\emptyset, \varphi_1)$, either $(\varphi_7 = \emptyset) \wedge (\varphi_8 = \varphi_1)$ or $\mathbf{Pub}(\mathbf{N}_\ell(\varphi_7, \varphi_8)) \wedge \mathbf{Taint}(\mathbf{N}_\ell(\emptyset, \varphi_1))$ holds. In the latter case, $\ell = \mathbf{Pub}$ and $\varphi_7 = \varphi_1 = \emptyset$. Thus, we have $\varphi_7 = \emptyset$ and $\varphi_8 \geq \varphi_1$ as required.
Since $(\varphi, \emptyset) \Longrightarrow_\Psi (\varphi_0 + \varphi_2 + \varphi_1', \{\mathbf{chk}_\ell(n, \varphi_2)\})$ for some $\varphi_1' \geq \varphi_8 \geq \varphi_1$, **ConsistentCap**$(E, \varphi', \Psi')$ follows from **ConsistentCap**(E, φ, Ψ).

– Case EXR-SPLT: In this case, $\Psi = \Psi_1 \uplus \{\mathbf{split}\ (V, W)\ \mathbf{is}\ (x, y).P\}$ and $\Psi' = \Psi_1 \uplus \{[V/x, W/y]P\}$, with $E' = E$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, and $\varphi' = \varphi$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l}
\Gamma; \varphi_1 \vdash_{\mathbf{ex}} \Psi_1 \\
\Gamma; \varphi_2 \vdash_{\mathbf{ex}} V : \tau_1 \\
\Gamma; \varphi_3 \vdash_{\mathbf{ex}} W : [V/0]\tau_2 \\
\Gamma, x : \tau_1, y : [x/0]\tau_2; \varphi_4 \vdash_{\mathbf{ex}} P \\
\varphi \geq \varphi_1 + \varphi_2 + \varphi_3 + \varphi_4
\end{array}$$

Here, φ_4 does not contain x and y. Without loss of generality, we also assume that x, y does not occur in V, W. By applying the substitution lemma (Lemma 10), we have $\Gamma, y : [V/0]\tau_2; \varphi_4 \vdash_{\mathbf{ex}} [V/x]P$. By applying the substitution lemma again, we get: $\Gamma; \varphi_4 \vdash_{\mathbf{ex}} [V/x, W/y]P$. Thus, we obtain $\Gamma'; \varphi' \vdash_{\mathbf{ex}} \Psi'$ as required. **ConsistentCap**(E, φ', Ψ') follows from **ConsistentCap**(E, φ, Ψ).

– Case EXR-MTCH: Similar to the case EXR-SPLT above.

– Case EXR-SDEC: Similar to the case EXR-ADEC below.

– Case EXR-ADEC: In this case, $\Psi = \Psi_1 \uplus \{\mathbf{decrypt}\ \{\!|V|\!\}_{k_1}\ \mathbf{is}\ \{\!|x|\!\}_{k_2^{-1}}.P\}$ and $\Psi' = \Psi_1 \uplus \{[V/x]P\}$, with $E' = E$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, and $\varphi' = \varphi$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l}
\Gamma; \varphi_1 \vdash_{\mathbf{ex}} \Psi_1 \\
\Gamma; \varphi_2 \vdash_{\mathbf{ex}} k_1 : \mathbf{EKey}(\tau_1) \\
\Gamma; \varphi_3 \vdash_{\mathbf{ex}} V : \tau_1 \\
\Gamma; \varphi_4 \vdash_{\mathbf{ex}} k_2 : \mathbf{DKey}(\tau_2) \\
\Gamma, x : \tau_2; \varphi_5 \vdash_{\mathbf{ex}} P \\
\varphi \geq \varphi_1 + \varphi_2 + \varphi_3 + \varphi_4 + \varphi_5 \\
\Gamma(k_1) = \mathbf{EKey}(\tau) \\
\Gamma(k_2) = \mathbf{DKey}(\tau)
\end{array}$$

By the 2nd, 4th, and the last two conditions, we have $\mathbf{EKey}(\tau) \leq_{\mathtt{ex}} \mathbf{EKey}(\tau_1)$ and $\mathbf{DKey}(\tau) \leq_{\mathtt{ex}} \mathbf{DKey}(\tau_2)$. $\mathbf{EKey}(\tau) \leq_{\mathtt{ex}} \mathbf{EKey}(\tau_1)$ implies $\tau = \tau_1$ or $\mathbf{Pub}(\mathbf{EKey}(\tau)) \wedge \mathbf{Taint}(\mathbf{EKey}(\tau_1))$, which implies, $\tau = \tau_1$ or $\mathbf{Taint}(\tau) \wedge \mathbf{Pub}(\tau_1)$. Thus, we have $\tau_1 \leq_{\mathtt{ex}} \tau$. Similarly, $\mathbf{DKey}(\tau) \leq_{\mathtt{ex}} \mathbf{DKey}(\tau_2)$ implies $\tau \leq_{\mathtt{ex}} \tau_2$. As the subtyping relation is transitive (Lemma 3), we have $\tau_1 \leq_{\mathtt{ex}} \tau_2$. Thus, by using EXT-SUB and the substitution lemma (Lemma 10), we obtain $\Gamma; \varphi_5 \vdash_{\mathbf{ex}} [V/x]P$. By EXT-WEAKCAP, we obtain $\Gamma; \varphi \vdash_{\mathbf{ex}} \Psi'$ as required.
ConsistentCap(E', φ', Ψ') follows immediately from **ConsistentCap**(E, φ, Ψ).

– Case ExR-Beg: In this case, $\Psi = \Psi_1 \uplus \{\mathbf{begin}\, V.P\}$ and $\Psi' = \Psi_1 \uplus \{P\}$, with $E' = E \uplus \{\mathbf{end}(V)\}$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, and $\varphi' = \varphi + \{\mathbf{end}(V) \mapsto 1\}$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l} \Gamma; \varphi_1 \vdash_{\mathbf{ex}} \Psi_1 \\ \Gamma; \varphi_2 + \{\mathbf{end}(V) \mapsto 1\} \vdash_{\mathbf{ex}} P \\ \varphi \geq \varphi_1 + \varphi_2 \end{array}$$

Thus, we have $\Gamma; \varphi' \vdash_{\mathbf{ex}} \Psi'$ as required. **ConsistentCap**(E', φ', Ψ') follows immediately from **ConsistentCap**(E, φ, Ψ).

– Case ExR-End: In this case, $\Psi = \Psi' \uplus \{\mathbf{end}\, V\}$ with $E = E' \uplus \{\mathbf{end}(V)\}$, $\Gamma' = \Gamma$, $\mathcal{K}' = \mathcal{K}$, and $\varphi = \varphi' + \{\mathbf{end}(V) \mapsto 1\}$. By the assumption $\vdash_{\mathbf{ex}} \langle \Psi, E, \Gamma, \mathcal{K}, \varphi \rangle$, we have:

$$\begin{array}{l} \Gamma; \varphi_1 \vdash_{\mathbf{ex}} \Psi' \\ \Gamma; \varphi_2 + \{\mathbf{end}(V) \mapsto 1\} \vdash_{\mathbf{ex}} \mathbf{end}\, V \\ \varphi \geq \varphi_1 + \varphi_2 + \{\mathbf{end}(V) \mapsto 1\} \end{array}$$

Thus, we have $\Gamma; \varphi \vdash_{\mathbf{ex}} \Psi'$ as required. **ConsistentCap**(E', φ', Ψ') follows immediately from **ConsistentCap**(E, φ, Ψ).

Lemma 2 now follows as an immediate corollary of the lemmas above.

Proof of Lemma 2 Suppose $\emptyset; \emptyset \vdash P$. By Lemma 5, there exists an extended process P' such that $\emptyset; \emptyset \vdash_{\mathbf{ex}} P'$ and $\mathbf{Erase}(P') = P$. By Lemma 7, $\langle P', \emptyset, \emptyset, \emptyset, \emptyset \rangle \not\longrightarrow^*_{\mathbf{ex}} \mathbf{Error}$. Thus, by Lemma 6 and $P = \mathbf{Erase}(P')$, P is safe. □

E Proof of Theorem 2

Lemma 12. *If $\Gamma, x : T \vdash_{\mathbf{GJ}} P$ and $x \notin fn(P)$ then $\Gamma \vdash_{\mathbf{GJ}} P$.*

Proof. Follows from Lemma 10 in the technical report for the Gordon-Jeffrey type system [13]

Lemma 13. *If $Public(T)$ then $\mathbf{Pub}([T])$. If $Tainted(T)$ then $\mathbf{Taint}([T])$.*

Proof. By straightforward induction in the derivation of $Public(T)$ and $Tainted(T)$ using their algorithmic formulation. Rules Tainted Top, Public Sum, Tainted Sum, Public Keypair, Tainted Keypair, and Public CR are not considered.

Lemma 14. *If $T \leq_{\mathbf{GJ}} \mathsf{Un}$ then $\mathbf{Pub}([T])$. If $\mathsf{Un} \leq_{\mathbf{GJ}} T$ then $\mathbf{Taint}([T])$.*

Proof. In both cases we see that rule Sub Public Tainted must have been used to derive the subtyping expression. In both cases Lemma 13 gives us the desired result.

Lemma 15. *If $\Gamma \vdash_{\mathbf{GJ}} M : T$ then $[\Gamma]; \emptyset \vdash [M] : [T]$.*

Proof. By straightforward induction in the derivation of $\Gamma \vdash_{\mathbf{GJ}} M : T$. Note that rules Msg Subsum, Msg Inl, Msg Inr, and Msg Part cannot happen by restriction.

Theorem 3. *If $\Gamma \vdash_{\mathbf{GJ}} P : es$ then $[\Gamma]; [es] \vdash [P]$.*

Proof. By induction in the derivation of $\Gamma \vdash_{\mathbf{GJ}} P : es$.

- Case PROC SUBSUM: by induction hypothesis and the fact that $es \leq es + fs$, we can apply rule T-CSUB to obtain the desired result.
- Case PROC OUTPUT UN: since $\Gamma \vdash_{\mathbf{GJ}} M : \mathsf{Un}$ we have by Lemma 15 that $[\Gamma]; \emptyset \vdash M : [\mathsf{Un}]$. As $[\mathsf{Un}] = \mathbf{N_{Pub}}(\emptyset, \emptyset)$ we get that the first condition for rule T-OUT is satisfied. Since $\Gamma \vdash_{\mathbf{GJ}} N : T$ we can again apply Lemma 15 to obtain $[\Gamma]; \emptyset \vdash N : [T]$ thereby satisfying the second condition for rule T-OUT. Finally, since $T \leq_{\mathbf{GJ}} \mathsf{Un}$ we get from Lemma 14 that $\mathbf{Pub}([T])$ and can then satisfy the final condition of rule T-OUT.
- Case PROC INPUT UN: since $\Gamma \vdash_{\mathbf{GJ}} M : \mathsf{Un}$ we have by Lemma 15 that $[\Gamma]; \emptyset \vdash M : [\mathsf{Un}]$. As $[\mathsf{Un}] = \mathbf{N_{Pub}}(\emptyset, \emptyset)$ we get that the first condition for rule T-IN is satisfied. Since $\Gamma, y : T \vdash_{\mathbf{GJ}} P : es$ we have by induction hypothesis that the second condition is satisfied. Finally, since $\mathsf{Un} \leq_{\mathbf{GJ}} T$ we get from Lemma 14 that $\mathbf{Taint}([T])$ and can then satisfy the final condition of rule T-IN.
- Case PROC REPEAT INPUT UN: similar to case PROC OUTPUT UN but also using rule T-REP.
- Case PROC PAR: by the induction hypothesis we can immediately apply rule T-PAR.
- Case PROC RES: we treat the different cases of T separately:
  - $T = \mathsf{Un}$: since $[\mathsf{Un}] = \mathbf{N_{Pub}}(\emptyset, \emptyset)$ we can apply the induction hypothesis and rule T-NEWN to obtain the desired result.
  - $T = \mathsf{SharedKey}(T')$: by the fact that $\mathbf{SKey}([T']) = [\mathsf{SharedKey}(T')]$ we can apply the induction hypothesis and rule T-NEWSK to obtain the desired result.
  - $T = \mathsf{KeyPair}(T')$: by the restricted used of KeyPair we know that two next constructions in P are **let** constructs follows by a process P'. By the typing of P we get that $E, x{:}\mathsf{KeyPair}(T'), y{:}\mathsf{Encrypt\ Key}(T'), z{:}\mathsf{Encrypt\ Key}(T') \vdash_{\mathbf{GJ}} P' : es$. By the fact that $x \notin fn(P')$ Lemma 12 gives us that $E, y{:}\mathsf{Encrypt\ Key}(T'), z{:}\mathsf{Encrypt\ Key}(T') \vdash_{\mathbf{GJ}} P' : es$. By induction hypothesis $[E], y{:}[\mathsf{Encrypt\ Key}(T')], z{:}[\mathsf{Encrypt\ Key}(T')]; [es] \vdash [P']$. Since $[\mathsf{Encrypt\ Key}(T')] = \mathbf{EKey}([T'])$ and $[\mathsf{Decrypt\ Key}(T')] = \mathbf{DKey}([T'])$ we can apply rule T-NEWAK to obtain the desired result.
- Case PROC SPLIT: follows by the induction hypothesis, Lemma 15, and rule T-SPLIT.
- Case PROC MATCH: follows by the induction hypothesis, Lemma 15, and rule T-MATCH.
- Case PROC CASE: cannot happen by our restrictions.
- Case PROC SYMM: follows by the induction hypothesis, Lemma 15, and rule T-SDEC.
- Case PROC ASYMM: follows by the induction hypothesis, Lemma 15, and rule T-ADEC.
- Case PROC BEGIN: follows by the induction hypothesis and rule T-BEGIN; if es does not contain an $\mathbf{end}(M)$ we have to extend it first using T-CSUB.
- Case PROC END: by our restrictions $P = \mathsf{end}\, L$; stop and hence the results follows by rule T-END.
- Case PROC WITNESS: cannot happen by our restrictions.
- Case PROC TRUST: cannot happen by our restrictions.

- Case PROC CAST: by assumption we have $\mathit{\Gamma} \vdash_{\mathbf{GJ}} x : l$ Challenge es_C and also $\mathit{\Gamma}, x : l$ Response $es_R \vdash_{\mathbf{GJ}} P : fs$. Lemma 15 then gives us that $[\mathit{\Gamma}]; \emptyset \vdash x :$ $[l$ Challenge $es_C]$ and the induction hypothesis that $[\mathit{\Gamma}], x : [l$ Response $es_R]; [fs] \vdash [P]$. Since $[l$ Challenge $es_C] = \mathbf{N}_l([es_C], \emptyset)$ we can apply rule T-NAME to obtain $[\mathit{\Gamma}]; [es_C] + [es_R] \vdash x : \mathbf{N}_l(\emptyset, [es_R])$. As $\mathbf{N}_l(\emptyset, [es_R]) = [l$ Response $es_R]$ we can apply rule T-LET to obtain the desired result.
- Case PROC CHECK: by assumption we have $\mathit{\Gamma} \vdash_{\mathbf{GJ}} M : l$ Challenge es_C, $\mathit{\Gamma} \vdash_{\mathbf{GJ}} N : l$ Response es_R, and $\mathit{\Gamma} \vdash_{\mathbf{GJ}} P : fs$. Lemma 15 then gives us that $[\mathit{\Gamma}]; \emptyset \vdash M :$ $[l$ Challenge $es_C]$ and $[\mathit{\Gamma}]; \emptyset \vdash N : [l$ Response $es_C]$. Since $[l$ Challenge $es_C] = \mathbf{N}_l([es_C], \emptyset)$ and $[l$ Response $es_R] = \mathbf{N}_l(\emptyset, [es_R])$ we can satisfy the two first premises of rule T-CHK using $\varphi_1 = \varphi_2 = \emptyset$. Now let $\varphi_3 = [es] = [fs] - ([es_C] + [es_R])$, $\varphi_4 = [es_C]$, and $\varphi_5 = [es_R]$. Since $\mathit{\Gamma} \vdash_{\mathbf{GJ}} P : fs$ we have by the induction hypothesis that $[\mathit{\Gamma}]; \varphi_3 + \varphi_4 + \varphi_5 \vdash [P]$ and we can finally apply rule T-CHK.
- Case PROC CHALLENGE: follows by induction hypothesis and rule T-RES.