Timed Spi-calculus with Types for Secrecy and Authenticity*

Christian Haack[1] and Alan Jeffrey[1,2]

[1] CTI, DePaul University
[2] Bell Labs, Lucent Technologies

Abstract. We present a discretely timed spi-calculus. A primitive for key compromise allows us to model key compromise attacks, thus going beyond the standard Dolev–Yao attacker model. A primitive for reading a global clock allows us to express protocols based on timestamps, which are common in practice. We accompany the timed spi-calculus with a type system, prove that well-typed protocols are robustly safe for secrecy and authenticity and present examples of well-typed protocols as well as an example where failure to typecheck reveals a (well-known) flaw.

1 Introduction

Models for cryptographic protocols often assume perfect cryptography— an example is the spi-calculus [3]— and ignore the fact that session keys can be compromised given a sufficient amount of time. Yet typical protocols for the distribution of session keys are careful to prevent attacks that fool honest agents into accepting compromised session keys. A security goal of such protocols is that after the end of a protocol run each principal possesses a session key that is currently secret (and will remain secret until its expiration time). This goal could not be expressed, for instance, in [11], which instead uses injective agreement as a security goal for key distribution protocols. In this paper, we extend the spi-calculus with a simple notion of time so that we can express such security goals. We also add a primitive for key compromise, which allows us to express key compromise attacks, thus going beyond the Dolev–Yao attacker model. A primitive for reading a global clock allows us to express protocols based on timestamps, which are common in practice.

Our model of time is very coarse and simple. A clock-tick represents the end of an epoch. Protocol designers may specify that a key is a short-term secret and a key compromise primitive cracks keys that are short-terms secrets. Cracking uses up all time of the current epoch (and not more than that) moving on to the next epoch. So after a clock-tick short-term secrets cannot be considered secret anymore and expire. Cracking a key is the only interesting action that uses up time. The usual spi-calculus actions are instantaneous. The safety of cryptographic protocols often depends on the fact that sessions expire when waiting for input for too long. We model this by letting input and most other statements expire with a clock-tick.

* This material is based upon work supported by the National Science Foundation under Grant No. 0208459.

We think that our simple model of time is enough to capture important aspects of security protocols in the presence of key compromise. On the other hand, because of its simplicity reasoning in this model remains tractable. In order to make this point, we have accompanied our timed spi-calculus with a type system for secrecy and authenticity and prove its robust safety. We show how an attempt to typecheck the Needham–Schroeder Symmetric Key Protocol [20] reveals its flaw and typecheck Denning–Sacco's fix [8] of this protocol. It turns out that proving our type system safe for short-term assertions is considerably simpler than the proofs for injective agreement in [11], which may suggest that short-term assertions are easier to reason about than injective agreement.

2 Syntax

Our protocol description language is an extension of the spi-calculus. In this section, we define its syntax: Messages are built from variables, time constants and the empty message by concatenation, symmetric encryption and message tagging. Unlike some other versions of the spi-calculus, we do not distinguish between variables and names. The ciphertext $\{M\}_K$ represents M encrypted with symmetric key K. Key K may be an arbitrary message, but the typing rules for honest agents require K to be a variable. The term $L(M)$ represents M tagged by label L. Label L may be an arbitrary message, but the typing rules for honest agents require L to be a variable. Message tagging is a common technique for avoiding type confusion attacks [17, 4] and is often treated explicitly in typed spi-calculi. A ciphertext that is formed by honest principals is typically of the form $\{l(M)\}_k$, where k is a secret key and l is a public message tag, whose purpose it is to distinguish the plaintext $l(M)$ from other plaintexts that are encrypted by the same key k.

Messages:

x, y, z, k, l, m, n	variables and names
$s, t \in \mathbb{N}$	discrete time
$K, L, M, N ::=$	message
x	variable or name
t	time
$()$	empty message
(M, N)	M concatenated with N
$\{M\}_K$	M encrypted with symmetric key K
$L(M)$	M tagged by L

As usual for spi-calculi, the process language includes a π-calculus extended with primitives for encryption. The importance of this paper is the inclusion of the operation $\mathsf{crack}\, M\, \mathsf{is}\, \{x{:}T\}_{y:U}$. This operation gives attackers the capability of cracking short-term keys given a sufficient amount of time. Thus, the attacker capabilities that we model go beyond the standard Dolev–Yao model. We also include an operation $\mathsf{clock}(x{:}T)$ for reading a global clock. This clock-operation permits to express protocols with timestamps, which are quite common in practice.

For specification purposes, secrecy and correspondence assertions may be inserted into programs. The meaning of secrecy assertions is the intuitive one. Correspondence

assertions are a standard method for specifying authenticity. They specify that in every protocol run every $\mathsf{end}(M)$-assertion must have been *recently* preceded by a corresponding $\mathsf{begin!}(M)$-assertion. In this paper, we restrict our attention to short-term, many-to-one correspondences for short-term, non-injective agreement.

Processes with Assertions:

$\vec{x}{:}\vec{T}$	type-annotated variables, $\lvert\vec{x}\rvert = \lvert\vec{T}\rvert$
$\tau \in \{\mathsf{lt}, \mathsf{st}\}$	long/short qualifier (long-term or short-term)
$O, P, Q, R ::=$	process
$\quad P \mid Q$	$\quad$ parallel composition
$\quad !P$	$\quad$ replication
$\quad \mathbf{0}$	$\quad$ inactivity
$\quad \mathsf{out}\ N\ M$	$\quad$ asynchronous output of message M on channel N
$\quad \pi; P$	$\quad$ prefix π followed by P
$\quad A$	$\quad$ assertion
$\pi ::=$	prefix
$\quad \mathsf{inp}\ N\ (x{:}T)$	$\quad$ input x from channel N (binding x in P)
$\quad \mathsf{new}(n{:}T)$	$\quad$ generating name n (binding n in T, P)
$\quad \mathsf{decrypt}\ M\ \mathsf{is}\ \{x{:}T\}_K$	$\quad$ decrypting M (binding x in P)
$\quad \mathsf{untag}\ M\ \mathsf{is}\ L(x{:}T)$	$\quad$ untagging M (binding x in P)
$\quad \mathsf{split}\ M\ \mathsf{is}\ (x{:}T, y{:}U)$	$\quad$ splitting M (binding x in U and x, y in P)
$\quad \mathsf{match}\ M\ \mathsf{is}\ (N, x{:}T)$	$\quad$ matching M against (N, x) (binding x in P)
$\quad \mathsf{crack}\ M\ \mathsf{is}\ \{x{:}T\}_{y{:}U}$	$\quad$ cracking key y of ciphertext M (binding x, y in P)
$\quad \mathsf{clock}(x{:}T)$	$\quad$ reading current time into x (binding x in P)
$\quad \mathsf{begin!}(M)$	$\quad$ short-term begin-assertion: begin session M
$A, B, C ::=$	assertions
$\quad \mathsf{end}(M)$	$\quad$ short-term end-assertion: session M has recently begun
$\quad \theta(M)$	$\quad$ secrecy assertion
$\theta ::=$	secrecy predicates
$\quad \tau\text{-}\mathsf{secret}$	$\quad$ $\tau = \mathsf{st}$: secret for the current epoch; $\tau = \mathsf{lt}$: secret forever
$\quad \mathsf{public}$	$\quad$ public

Prefixing and replication bind more tightly than parallel composition. We often elide $\mathbf{0}$ from the end of processes, write $(\mathsf{out}\ N\ M; P)$ for $(\mathsf{out}\ N\ M \mid P)$, write $(A; P)$ for $(A \mid P)$, and write $(\mathsf{new}\ \theta\ (n{:}T); P)$ for $(\mathsf{new}(n{:}T); \theta(n); P)$. We write $fv(P)$ for the set of free variables of P; similarly for messages and other objects that may contain variables.

3 Semantics

The architecture of our operational semantics is inspired by [18]. It is defined as a reduction relation on states of the form $(\underline{t}; \vec{n}; \overline{A} \mid\mid P)$, where $\underline{t}$ is a natural number representing the global time, $\vec{n}$ is a binder for (A, P)'s free names, $\overline{A}$ is the set of correspondences that can be ended in the particular run, and P is the process that remains to be executed. The reduction rules are divided into a set of *instantaneous reductions*, which are assumed to take no time, and a set of *tick-reductions*, which use up all time in the current epoch moving on to the next epoch. The instantaneous reductions are pretty standard:

Structural Process Equivalence, $P \equiv Q$:

$P \equiv P$	(Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$Q \equiv R \Rightarrow P \mid Q \equiv P \mid R$	(Struct Par)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)

Instantaneous Reductions, $(t;\vec{n};\overline{A} \mid\mid P) \rightarrow (t;\vec{m};\overline{B} \mid\mid Q)$:

$P \equiv P', (t;\vec{n};\overline{A} \mid\mid P') \rightarrow (t;\vec{m};\overline{B} \mid\mid Q'), Q' \equiv Q \;\Rightarrow\; (t;\vec{n};\overline{A} \mid\mid P) \rightarrow (t;\vec{m};\overline{B} \mid\mid Q)$	(Redn Equiv)
$m \notin \underline{fv}(\vec{n},Q) \;\Rightarrow\; (t;\vec{n};\overline{A} \mid\mid \mathsf{new}(m{:}T);P \mid Q) \rightarrow (t;\vec{n},m;\overline{A} \mid\mid P \mid Q)$	(Redn New)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{out}\ N\ M \mid \mathsf{inp}\ N\ (x{:}T);P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x{\leftarrow}M\} \mid Q)$	(Redn IO)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{decrypt}\ \{M\}_K\ \mathsf{is}\ \{x{:}T\}_K;P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x{\leftarrow}M\} \mid Q)$	(Redn Decrypt)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{untag}\ L(M)\ \mathsf{is}\ L(x{:}T);P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x{\leftarrow}M\} \mid Q)$	(Redn Untag)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{split}\ (M,N)\ \mathsf{is}\ (x{:}T,y{:}U);P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x,y{\leftarrow}M,N\} \mid Q)$	(Redn Split)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{match}\ (M,N)\ \mathsf{is}\ (M,x{:}T);P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x{\leftarrow}N\} \mid Q)$	(Redn Match)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{clock}(x{:}T);P \mid Q) \rightarrow (t;\vec{n};\underline{A} \mid\mid P\{x{\leftarrow}t\} \mid Q)$	(Redn Clock)
$(t;\vec{n};\underline{A} \mid\mid \mathsf{begin!}(M);P \mid Q) \rightarrow (t;\vec{n};\underline{A},\mathsf{end}(M) \mid\mid P \mid Q)$	(Redn Begin)

Tick-Reductions, $(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid Q)$:

(Tick Par)

$$\frac{(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid P') \quad (t;\vec{n};\overline{A} \mid\mid Q) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid Q')}{(t;\vec{n};\overline{A} \mid\mid P \mid Q) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid P' \mid Q')}$$

(Tick Crack)

$$\frac{P = (\mathsf{st\text{-}secret}(K) \mid \mathsf{crack}\ \{M\}_K\ \mathsf{is}\ \{x{:}T\}_{y:U};Q)}{(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid Q\{x,y{\leftarrow}M,K\})}$$

(Tick Remain)

$$\frac{P \text{ is } !Q, (\mathsf{out}\ N\ M), \mathsf{public}(M) \text{ or } \mathsf{lt\text{-}secret}(M)}{(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid P)}$$

(Tick Expire)

$$\frac{}{(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid \mathbf{0})}$$

We write $\Rightarrow$ for the reflexive and transitive closure of $(\rightarrow \cup \xrightarrow{\sigma})$.

The tick-reduction rule (Tick Par) ensures that a clock-tick happens simultaneously in every branch of a parallel composition. crack operates as expected and uses up time. Process replication and output remain alive in the next epoch. Importantly, every other syntactic form expires with a clock-tick, degenerating to the null-process. In particular, if a process waits for input for more than one epoch it aborts and declines to accept later incoming messages. Many security protocols depend on this kind of behavior, and we have decided to make *expiring input* the default in our process calculus. Because

process replication survives clock-ticks, we can express the capability to start a session at any time in the future. Our choice to let asynchronous output survive clock-ticks is a bit arbitrary. A language where output expires would probably have been equally suitable for modeling security protocols.

Definition 1 (Safety). P is *safe for secrecy* iff $(s; fv(P); \emptyset \mid\mid P) \not\Rightarrow (t; \vec{n}; \overline{A} \mid\mid \mathsf{public}(N) \mid \tau\text{-}\mathsf{secret}(M) \mid \mathsf{out}\, N\, M \mid Q)$. P is *safe for authenticity* iff $(s; fv(P); \emptyset \mid\mid P) \Rightarrow (t; \vec{n}; \overline{A} \mid\mid \mathsf{end}(M) \mid Q)$ implies $\mathsf{end}(M) \in A$. P is *safe* iff it is both safe for secrecy and authenticity.

Definition 2 (Opponent Processes). A process is an *opponent process* iff its only assertions are of the form $\mathsf{public}(M)$ and all its type annotations are the special type Un.

Definition 3 (Robust Safety). A process P is *robustly safe* iff $(P \mid O)$ is safe for all opponent processes O.

Our type system is designed so that well-typed processes with public free names are robustly safe:

Theorem (Robust Safety) *If* $(\vec{n}{:}\vec{T} \vdash \mathsf{public}(\vec{n}))$ *and* $(\vec{n}{:}\vec{T} \vdash P)$*, then P is robustly safe.*

4 Examples

We will use derived forms for lists and matching against tagged lists. Their definition uses derived forms for list types as type annotations. Type annotations have no impact operationally and the definition of list types is postponed to the type system.

Derived Forms for Lists and Matching Against Lists:

$\langle\rangle \triangleq ()$ $\quad \langle M\rangle \triangleq (M, ())$ $\quad \langle M, \vec{N}\rangle \triangleq (M, \langle\vec{N}\rangle)$

$\mathsf{match}\ M\ \mathsf{is}\ \langle N\rangle[\overline{A}] \triangleq \mathsf{match}\ M\ \mathsf{is}\ (N, x{:}\langle\rangle[\overline{A}])$

$\mathsf{match}\ M\ \mathsf{is}\ \langle x{:}T\rangle[\overline{A}] \triangleq \mathsf{split}\ M\ \mathsf{is}\ (x{:}T, y{:}\langle\rangle[\overline{A}])$

$\mathsf{match}\ M\ \mathsf{is}\ \langle x{:}T, nxts\rangle[\overline{A}] \triangleq \mathsf{split}\ M\ \mathsf{is}\ (x{:}T, y{:}\langle nxts\rangle[\overline{A}]);\ \mathsf{match}\ y\ \mathsf{is}\ \langle nxts\rangle[\overline{A}]$

$\mathsf{match}\ M\ \mathsf{is}\ \langle N, nxts\rangle[\overline{A}] \triangleq \mathsf{match}\ M\ \mathsf{is}\ (N, y{:}\langle nxts\rangle[\overline{A}]);\ \mathsf{match}\ y\ \mathsf{is}\ \langle nxts\rangle[\overline{A}]$

$\mathsf{match}\ M\ \mathsf{is}\ L\langle nxts\rangle[\overline{A}] \triangleq \mathsf{untag}\ M\ \mathsf{is}\ L(x{:}\langle nxts\rangle[\overline{A}]);\ \mathsf{match}\ x\ \mathsf{is}\ \langle nxts\rangle[\overline{A}]$

Example 1: Establishing a session key using a nonce.

B generates nonce n
$B \to A \quad n$
A generates short-term secrets *kab* and m
A begins! "A sending session key *kab* to B" and "A sending secret message m to B"
$A \to B \quad \{msg_1\langle n, kab\rangle\}_{lab}, \{msg_2\langle m\rangle\}_{kab}$
B asserts $\mathsf{st\text{-}secret}(kab)$ and ends "A sending session key *kab* to B"
B asserts $\mathsf{st\text{-}secret}(m)$ and ends "A sending secret message m to B"

Bob wants to receive a secret message from Alice. To this end, he sends Alice a freshly generated nonce n. In reply, Alice generates a short-term session key *kab* and sends it to Bob together with n and encrypted by their shared long-term key *lab*. Alice also sends the secret message m encrypted by *kab*. The names msg_1 and msg_2 are used as tags.

To express this protocol in spi, we assume that X is some finite set of principal names and abbreviate $\mathsf{new}_{a,b\in X}$ lt-secret $(lab{:}?)$ for the generation of their long-term keys and $\prod_{x,y\in X} P(x,y)$ for the parallel composition of all processes $P(x,y)$. We use the additional tags *key* and *sec* in our correspondence assertions.

$$
\begin{array}{l}
P \stackrel{\Delta}{=} \mathsf{public}(net) \mid \mathsf{new}_{a,b\in X}\ \mathsf{lt\text{-}secret}\ (lab{:}?);\ \prod_{a,b\in X}(!P_A(a,b,lab) \mid !P_B(a,b,lab)) \\
P_A(a{:}?,b{:}?,lab{:}?) \stackrel{\Delta}{=} \\
\quad \mathsf{inp}\ net\ (n{:}?);\ \mathsf{new\ st\text{-}secret}\ (kab{:}?);\ \mathsf{begin!}(key(a,kab,b));\ \mathsf{new\ st\text{-}secret}\ (m{:}?); \\
\quad \mathsf{begin!}(sec(a,m,b)); \mathsf{out}\ net\ (\{msg_1\langle n,kab\rangle\}_{lab}, \{msg_2\langle m\rangle\}_{kab}) \\
P_B(a{:}?,b{:}?,lab{:}?) \stackrel{\Delta}{=} \\
\quad \mathsf{new\ public}\ (n{:}?);\ \mathsf{out}\ net\ n;\ \mathsf{inp}\ net\ (x{:}?,u{:}?);\ \mathsf{decrypt}\ x\ \mathsf{is}\ \{y{:}?\}_{lab}; \\
\quad \mathsf{match}\ y\ \mathsf{is}\ msg_1\langle n,kab{:}?\rangle[?];\ \mathsf{st\text{-}secret}(kab);\ \mathsf{end}(key(a,kab,b)); \\
\quad \mathsf{decrypt}\ u\ \mathsf{is}\ \{v{:}?\}_{kab};\ \mathsf{match}\ v\ \mathsf{is}\ msg_2\langle m{:}?\rangle[?];\ \mathsf{st\text{-}secret}(m);\ \mathsf{end}(sec(a,m,b))
\end{array}
$$

This protocol is robustly safe: Bob only accepts the session key *kab* if received shortly after he generated nonce *n*. Because the ciphertext contains the fresh nonce, Bob knows that it must have been formed recently and that it is not a replay of an old message. Consequently, the session key that is contained in the ciphertext is still a secret and A has recently begun the $key(A,kab,B)$-session. Bob's second secrecy and end-assertions are safe, because Bob's session expires before opponents can possibly have cracked the session key. With appropriate type annotations this protocol typechecks.

Example 2: Needham–Schroeder Symmetric Key Protocol (NSSK). In this protocol, Alice and Bob want to establish a short-term session key *kab* via key server S using long-term keys *las* and *lbs*. NSSK is not robustly safe and, by the robust safety theorem, does not typecheck.

$$
\begin{array}{ll}
A \text{ generates nonce } na & \\
A \to S & A,B,na \\
S \text{ generates short-term secret } kab & \\
S \text{ begins! } init(kab,A,B) \text{ and } resp(kab,B,A) & \\
S \to A & \{msg_2\langle na,B,kab,\{msg_3\langle A,kab\rangle\}_{lbs}\rangle\}_{las} \\
A \text{ asserts } \mathsf{st\text{-}secret}(kab) \text{ and ends } init(kab,A,B) & \\
A \to B & \{msg_3\langle A,kab\rangle\}_{lbs} \\
B \text{ generates nonce } nb & \\
B \to A & \{msg_4\langle nb\rangle\}_{kab} \\
A \to B & \{msg_5\langle nb\rangle\}_{kab} \\
B \text{ asserts } \mathsf{st\text{-}secret}(kab) \text{ and ends } resp(kab,B,A) &
\end{array}
$$

Alice's secrecy- and end-assertions are safe. Bob's secrecy and end-assertions, however, are unsafe. The problem is that msg_3 may be a replay from an old protocol run. Here is an opponent process O that compromises this protocol; $(NSSK \mid O)$ is unsafe for both secrecy and authenticity:

$$
\begin{array}{ll}
O \stackrel{\Delta}{=} \mathsf{inp}\ net\ (m_3{:}\mathsf{Un});\ \mathsf{out}\ net\ m_3; & // \text{ monitoring } msg_3 \\
\quad \mathsf{inp}\ net\ (m_4{:}\mathsf{Un});\ \mathsf{out}\ net\ m_4; & // \text{ monitoring } msg_4 \\
\quad \mathsf{crack}\ m_4\ \mathsf{is}\ \{x{:}\mathsf{Un}\}_{kab:\mathsf{Un}}; & // \text{ cracking short-term key } kab \\
\quad \mathsf{inp}\ net\ (m_3'{:}\mathsf{Un}); & // \text{ intercepting } msg_3 \text{ from a later protocol run} \\
\quad \mathsf{out}\ net\ m_3; & // \text{ sending } m_3 \text{ instead of } m_3' \text{ to Bob} \\
\quad \mathsf{out}\ net\ kab & // \text{ publishing old key } kab
\end{array}
$$

The output statement $(\mathsf{out}\ net\ m_3)$ results in a violation of Bob's end-assertion, because Bob wants to end an old $resp(kab,B,A)$-session, but is only entitled to end a $resp(kab',B,A)$-session, where kab' is the new session key that is contained in message m'_3. The output statement $(\mathsf{out}\ net\ kab)$ obviously violates Bob's secrecy assertion $\mathsf{st\text{-}secret}(kab)$.

5 Type System

For the type system we extend the set of assertions from Section 2:

Type-Level Assertions:

$A,B,C ::=$	assertions
$\ldots$	as defined in Section 2
$M : T$	M has type T
$\mathsf{fresh}(N)$	N is a fresh nonce
$\mathsf{now}(N)$	N is the current time
$N\text{-}\mathsf{stamped}_\mathsf{t}(A)$	A is stamped by time N
$N\text{-}\mathsf{stamped}_\mathsf{n}(A)$	A is stamped by nonce N

Type-level assertions are needed to define type environments: An *environment* is simply an assertion set. Let $E,F,G,\bar{A},\bar{B},\bar{C}$ range over *environments*. An important judgment of our system is *assertion entailment*, $E \vdash \bar{A}$. We usually use meta-variables E,F,G left of $\vdash$ and $\bar{A},\bar{B},\bar{C}$ right of $\vdash$. We define the *subjects* of environments: $subj(\emptyset) \triangleq \emptyset$; $subj(E,M{:}T) \triangleq subj(E) \cup \{M\}$; $subj(E,A) \triangleq subj(E)$ otherwise. Let $(E \vdash \diamond)$ iff $fv(E) \subseteq fv(subj(E))$. We often write $(M_1,\ldots,M_n){:}(T_1,\ldots,T_n)$ for $\{M_1{:}T_1,\ldots,M_n{:}T_n\}$, write $\vec{M}{:}T$ for $\{M{:}T \mid M \in \vec{M}\}$, write $\theta\{\vec{M}\}$ for $\{\theta(M) \mid M \in \vec{M}\}$, write $\mathsf{end}\{\vec{M}\}$ for $\{\mathsf{end}(M) \mid M \in \vec{M}\}$, write $\mathsf{fresh}\{\vec{N}\}$ for $\{\mathsf{fresh}(N) \mid N \in \vec{N}\}$, and write $N\text{-}\mathsf{stamped}_i(\bar{A})$ for $\{N\text{-}\mathsf{stamped}_i(A) \mid A \in \bar{A}\}$ if $i \in \{\mathsf{t},\mathsf{n}\}$.

Types:

$T,U,V,W ::=$	types
Top	well-typed text
Un	public text
$\tau\text{-}\mathsf{Secret}$	τ-secret
$\tau\text{-}\mathsf{Key}(\vec{M})$	principals $\vec{M}$'s shared τ-key
$\mathsf{Tag}(X)$	tag of type-scheme X
$\tau\text{-}\mathsf{Auth}(K,\vec{M})$	plaintext to be authenticated by principals $\vec{M}$'s shared τ-key K
$(x{:}T,U)$	T-text paired with U-text (binding x in U)
$\mathsf{Ok}(\bar{A})$	empty text with precondition $\bar{A}$
$X,Y,Z ::=$	type-schemes for tags
$T \to \tau\text{-}\mathsf{Auth}(k{:}U,\vec{x}{:}\vec{V})$	text T to tagged text $\tau\text{-}\mathsf{Auth}(k,\vec{x})$ (binding $k,\vec{x}$ in $T,U,\vec{V}$)

A type T is called *generative* iff $T = \mathsf{Ok}(\bar{A})$ implies $\bar{A} = \emptyset$. Names generated by new are required to have generative types.

Types include a top type, dependent pair types, a type Un for public messages and types τ-Secret (where $\tau \in \{\mathsf{lt},\mathsf{st}\}$) for long- or short-term secrets. In addition, there are the following types:

– The *key type* $\tau\text{-}\mathsf{Key}(\vec{M})$ is the type of secret keys shared by principals $\vec{M}$.
– The *ok-type* $\mathsf{Ok}(\bar{A})$ is a type for the empty message. In order to assign this type to the empty message in environment E, it is required that $(E \vdash \bar{A})$.
– The *authentication type* $\tau\text{-}\mathsf{Auth}(K,\vec{M})$ is a type of tagged messages that require authentication by principals $\vec{M}$'s shared τ-key K.
– The *tag type* $\mathsf{Tag}(T \to \tau\text{-}\mathsf{Auth}(k{:}U,\vec{x}{:}\vec{V}))$ is a type of tags l that may tag messages M of type $T\{k,\vec{x}{\leftarrow}K,\vec{N}\}$. The type of the resulting tagged message $l(M)$ is $\tau\text{-}\mathsf{Auth}(K,\vec{N})$.

For instance, consider the following tag:

$$l \;:\; \mathsf{Tag}(\;(x{:}\mathsf{st\text{-}Secret}, \mathsf{Ok}(\mathsf{end}(\mathit{sec}(p,x,q)))) \to \mathsf{st\text{-}Auth}(k{:}\mathsf{st\text{-}Key}(p,q), p{:}\mathsf{Un}, q{:}\mathsf{Un})\;)$$

In environment $E = (A{:}\mathsf{Un}, B{:}\mathsf{Un}, \mathit{kab}{:}\mathsf{st\text{-}Key}(A,B), m{:}\mathsf{st\text{-}Secret}, \mathsf{end}(\mathit{sec}(A,m,B)))$, this tag can be used to tag message $\langle m\rangle$ $(= (m,()))$ forming $l\langle m\rangle$ of type $\mathsf{st\text{-}Auth}(\mathit{kab},A,B)$, which can then be authenticated by encryption with *kab* resulting in $\{l\langle m\rangle\}_{\mathit{kab}}$. Type-schemes for tags are a form of dependent types. Technically, they resemble type-schemes for polymorphic data constructors in languages like Haskell or ML (with the difference that binders range over messages instead of types).

Ok-types are important as a tool to "statically communicate" assertions between parallel processes for the purpose of typechecking. Typically, the set $\bar{A}$ in $\mathsf{Ok}(\bar{A})$ contains assertions of the form $\mathsf{end}(M)$ indicating that it is safe to end M-sessions. When typechecking a *sender* of the empty message at type $\mathsf{Ok}(\mathsf{end}(M))$, the typechecker is *required to prove* that it is safe to end M-sessions. On the other hand, when typechecking a *receiver* of a message of type $\mathsf{Ok}(\mathsf{end}(M))$, the typechecker *may use* that it is safe to end M-sessions.

Subtyping, $T \le U$:

$$\frac{}{T \le T}\;(\text{Sub Refl}) \qquad \frac{}{T \le \mathsf{Top}}\;(\text{Sub Top}) \qquad \frac{}{\tau\text{-}\mathsf{Key}(\vec{M}) \le \tau\text{-}\mathsf{Secret}}\;(\text{Sub Key}) \qquad \frac{}{\mathsf{Tag}(X) \le \mathsf{Un}}\;(\text{Sub Tag})$$

$$\frac{T \le T' \quad U \le U'}{(x{:}T,U) \le (x{:}T',U')}\;(\text{Sub Pair}) \qquad \frac{T \le \mathsf{Un} \quad U \le \mathsf{Un}}{(x{:}T,U) \le \mathsf{Un}}\;(\text{Sub Pair Un}) \qquad \frac{}{\mathsf{Ok}() \le \mathsf{Un}}\;(\text{Sub Ok Un})$$

$$\frac{E \vdash \diamond \quad \mathit{fv}(T,U) \subseteq \mathit{fv}(E) \quad T \le U}{E \vdash T \le U}\;(\text{Sub Env})$$

The rule (Sub Key) expresses that long- or short-term keys are long- or short-term secrets, and (Sub Tag) expresses that tags are public. Pair types are covariant, by (Sub Pair). The rules (Sub Ok Un) and (Sub Pair Un) express that the empty message and pairs of public messages may be published.

Step-Function, $\mathit{step}(\bar{A})$:

$\mathit{step}(T) \triangleq \mathsf{Un}$, if $T \le \mathsf{st\text{-}Secret}$ or $T = \mathsf{st\text{-}Auth}(K,\vec{M})$; $\mathit{step}(\mathsf{Ok}(\bar{A})) \triangleq \mathsf{Ok}(\mathit{step}(\bar{A}))$;
$\mathit{step}(x{:}T,U) \triangleq (x{:}\mathit{step}(T),\mathit{step}(U))$; $\mathit{step}(T) \triangleq T$, otherwise; $\mathit{step}(\mathsf{end}(M)) \triangleq \emptyset$;
$\mathit{step}(\mathsf{st\text{-}secret}(M)) \triangleq \{\mathsf{public}(M)\}$; $\mathit{step}(\mathsf{fresh}(N)) \triangleq \mathit{step}(\mathsf{now}(N)) \triangleq \emptyset$;
$\mathit{step}(M{:}T) \triangleq \{M{:}\mathit{step}(T)\}$; $\mathit{step}(A) \triangleq \{A\}$, otherwise; $\mathit{step}(\bar{A}) \triangleq \cup\{\mathit{step}(A) \mid A \in \bar{A}\}$

The *step*-function maps an assertion set to the assertion set that it evolves into with a clock-tick: assertions $\mathsf{fresh}(N)$ or $\mathsf{now}(N)$ are dropped, $\mathsf{st\text{-}secret}(M)$ is mapped to $\mathsf{public}(M)$, short-term types are mapped to Un, and all other clauses are the identity or defined by structural induction. We call an assertion set $\bar{A}$ *long-term* if $\mathit{step}(\bar{A}) = \bar{A}$, and *short-term* otherwise.

Assertion Entailment, $E \vdash \bar{A}$:

(Id) $\dfrac{E, A \vdash \diamond}{E, A \vdash A}$ (And) $\dfrac{E \vdash A_1 \cdots E \vdash A_n}{E \vdash A_1, \ldots, A_n}$ (Public) $\dfrac{E \vdash M : \mathsf{Un}}{E \vdash \mathsf{public}(M)}$ (Secret) $\dfrac{E \vdash M : \tau\text{-}\mathsf{Secret}}{E \vdash \tau\text{-}\mathsf{secret}(M)}$ (Time) $\dfrac{E \vdash \diamond}{E \vdash t : \mathsf{Un}}$

(Nonce Stamp) $\dfrac{E \vdash N : \mathsf{Top}, A}{E \vdash N\text{-}\mathsf{stamped}_{\mathsf{n}}(A)}$ (Time Stamp) $\dfrac{E \vdash N : \mathsf{Top}, \mathsf{now}(N), A}{E \vdash N\text{-}\mathsf{stamped}_{\mathsf{t}}(A)}$ (Sub) $\dfrac{E \vdash M : T \quad E \vdash T \leq U}{E \vdash M : U}$

(Encrypt) $\dfrac{E \vdash K : \tau\text{-}\mathsf{Key}(\vec{N}), M : \tau\text{-}\mathsf{Auth}(K, \vec{N})}{E \vdash \{M\}_K : \mathsf{Un}}$ (Encrypt Un) $\dfrac{E \vdash K : \mathsf{Un}, M : \mathsf{Un}}{E \vdash \{M\}_K : \mathsf{Un}}$ (Tag Un) $\dfrac{E \vdash L : \mathsf{Un}, M : \mathsf{Un}}{E \vdash L(M) : \mathsf{Un}}$

(Tag) $\rho = (k, \vec{x} \leftarrow K, \vec{N})$
$\tau = \mathsf{st} \Rightarrow \mathit{step}(T) \leq \mathsf{Un} \quad \tau = \mathsf{lt} \Rightarrow \mathit{step}(T, U, \vec{V}) = (T, U, \vec{V})$
$$\frac{E \vdash L : \mathsf{Tag}(T \rightarrow \tau\text{-}\mathsf{Auth}(k{:}U, \vec{x}{:}\vec{V})), M{:}T\{\rho\}, K{:}U\{\rho\}, \vec{N}{:}\vec{V}\{\rho\}}{E \vdash L(M) : \tau\text{-}\mathsf{Auth}(K, \vec{N})}$$

(Pair) $\dfrac{E \vdash M : T, N : U\{x \leftarrow M\}}{E \vdash (M, N) : (x{:}T, U)}$ (Empty) $\dfrac{E \vdash \bar{A}}{E \vdash () : \mathsf{Ok}(\bar{A})}$

The rule (Time) expresses that time values are public. (Nonce Stamp) is typically used to stamp short-term assertions. Importantly, stamped assertions are long-term. Thus, the rule (Nonce Stamp) turns short-term assertions into long-term assertions by associating them with a nonce N. The rule (Time Stamp) is similar. The process-level typing rules (Nonce Unstamp) and (Time Unstamp) presented below, then permit a "receiver" of an assertion $N\text{-}\mathsf{stamped}_i(A)$ to use A, if he can validate that N is a fresh nonce or the current time. Note that the rule (Time Stamp) requires that the "creator" of a time-stamped assertion knows that the stamp is current: it would be dangerous if he used a future timestamp. The rule (Encrypt) is consistent with our informal interpretation of authentication types; ciphertexts are public. (Encrypt Un) and (Tag Un) allow us to type-check Dolev–Yao attackers; typically, these rules are not used for type-checking honest agents. Perhaps the most interesting typing rule is (Tag) for formation of trusted tagged messages. The premise for $\tau = \mathsf{st}$ enforces that short-term keys may not encrypt long-term secrets; a requirement that is obviously needed for long-term secrecy. The premise for $\tau = \mathsf{lt}$ enforces that long-term keys may not encrypt messages of short-term types. Without this premise the system would be unsafe because a receiver of a short-term assertion under a long-term key has no guarantee that the short-term assertion is still

valid at the time of reception. It is still possible to communicate short-term assertions under long-term keys, provided the short-term assertions are associated with nonces or timestamps using (Nonce Stamp) and (Time Stamp). The rule (Pair) is the standard rule for dependent pair types.

Well-Typed Processes, $E \vdash P$**:**

(Par) $\dfrac{E \vdash P \quad E \vdash Q}{E \vdash P \mid Q}$ (Repl) $\dfrac{E \vdash P \quad \mathit{step}(E) \vdash P}{E \vdash !P}$ (Zero) $\dfrac{E \vdash \diamond}{E \vdash \mathbf{0}}$ (Begin) $\dfrac{E, \mathsf{end}(M) \vdash P}{E \vdash \mathsf{begin!}(M); P}$

(Out) $\dfrac{E \vdash N : \mathsf{Un}, M : \mathsf{Un}}{E \vdash \mathsf{out}\ N\ M}$ (In) $x \notin \mathit{fv}(E)$ $\dfrac{E \vdash N : \mathsf{Un} \quad E, x : \mathsf{Un} \vdash P}{E \vdash \mathsf{inp}\ N\ (x{:}\mathsf{Un}); P}$ (New) $n \notin \mathit{fv}(E)$, T generative $\dfrac{E, n : T, \mathsf{fresh}(n) \vdash P}{E \vdash \mathsf{new}(n{:}T); P}$

(Clock) $x \notin \mathit{fv}(E)$ $\dfrac{E, x : \mathsf{Un}, \mathsf{now}(x) \vdash P}{E \vdash \mathsf{clock}(x{:}\mathsf{Un}); P}$ (Crack) $x,y \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : \mathsf{Un} \quad \mathit{step}(E), x : \mathsf{Un}, y : \mathsf{Un} \vdash P}{E \vdash \mathsf{crack}\ M\ \mathsf{is}\ \{x{:}\mathsf{Un}\}_{y:\mathsf{Un}}; P}$

(Decrypt) $x \notin \mathit{fv}(E)$
$$\frac{E \vdash M : \mathsf{Un}, K : \tau\text{-}\mathsf{Key}(\vec{N}) \quad E, x : \tau\text{-}\mathsf{Auth}(K, \vec{N}) \vdash P}{E \vdash \mathsf{decrypt}\ M\ \mathsf{is}\ \{x : \tau\text{-}\mathsf{Auth}(K, \vec{N})\}_K; P}$$

(Decrypt Un) $x \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : \mathsf{Un}, K : \mathsf{Un} \quad E, x : \mathsf{Un} \vdash P}{E \vdash \mathsf{decrypt}\ M\ \mathsf{is}\ \{x{:}\mathsf{Un}\}_K; P}$ (Untag Un) $x \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : \mathsf{Un}, L : \mathsf{Un} \quad E, x : \mathsf{Un} \vdash P}{E \vdash \mathsf{untag}\ M\ \mathsf{is}\ L(x{:}\mathsf{Un}); P}$

(Untag) $x \notin \mathit{fv}(E) \quad \rho = (k, \vec{y} \leftarrow K, \vec{N}) \quad E \vdash T\{\rho\} \leq U$
$$\frac{E \vdash M : \tau\text{-}\mathsf{Auth}(K, \vec{N}), L : \mathsf{Tag}(T \to \tau\text{-}\mathsf{Auth}(k{:}U, \vec{y}{:}\vec{V})) \quad E, x : U \vdash P}{E \vdash \mathsf{untag}\ M\ \mathsf{is}\ L(x{:}U); P}$$

(Split) $x,y \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : (x{:}T, U) \quad E, x : T, y : U \vdash P}{E \vdash \mathsf{split}\ M\ \mathsf{is}\ (x{:}T, y{:}U); P}$ (Split Un) $x,y \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : \mathsf{Un} \quad E, x : \mathsf{Un}, y : \mathsf{Un} \vdash P}{E \vdash \mathsf{split}\ M\ \mathsf{is}\ (x{:}\mathsf{Un}, y{:}\mathsf{Un}); P}$

(Match) $y \notin \mathit{fv}(E) \quad \rho = (x \leftarrow N)$ $\dfrac{E \vdash M : (x{:}\mathsf{Top}, T), N : \mathsf{Top} \quad E, y : T\{\rho\} \vdash P}{E \vdash \mathsf{match}\ M\ \mathsf{is}\ (N, y{:}T\{\rho\}); P}$ (Match Un) $x \notin \mathit{fv}(E)$ $\dfrac{E \vdash M : \mathsf{Un}, N : \mathsf{Un} \quad E, x : \mathsf{Un} \vdash P}{E \vdash \mathsf{match}\ M\ \mathsf{is}\ (N, x{:}\mathsf{Un}); P}$

(Nonce Unstamp) $\dfrac{E, A \vdash P}{E \vdash \mathsf{fresh}(N), N\text{-}\mathsf{stamped}_\mathsf{n}(A)}$ $\dfrac{}{E \vdash P}$ (Time Unstamp) $\dfrac{E, A \vdash P}{E \vdash \mathsf{now}(N), N\text{-}\mathsf{stamped}_\mathsf{t}(A)}$ $\dfrac{}{E \vdash P}$ (Ok) $\dfrac{E, \overline{A} \vdash P}{E \vdash M : \mathsf{Ok}(\overline{A})}$ $\dfrac{}{E \vdash P}$

Among the process rules, (Repl) for process replication is noteworthy, because it requires to typecheck the body P of a replicated process $!P$ both in the current environment E and the future environment $\mathit{step}(E)$. Checking P in E is needed because

replicated processes unfold instantaneously (by (Redn Equiv)); checking P in $\mathit{step}(E)$ is needed because replicated processes survive clock-ticks (by (Tick Remain)); because the *step*-function is idempotent, it suffices to check P in environment $\mathit{step}(E)$ instead of $\mathit{step}^n(E)$ for all $n \geq 1$. For typechecking the process continuation P in $(\mathsf{new}(x{:}T);P)$ or $(\mathsf{clock}(x{:}\mathsf{Un});P)$, we may assume that x is fresh or current. Remember that specification processes $\mathsf{end}(M)$ and $\theta(M)$ are both processes and assertions, so their typing rules are given with the rules for assertion entailment.

6 Typed Examples

We will annotate the earlier example with types using these derived forms:

Derived Forms for List Types:

$$\langle\rangle[\bar{A}] \triangleq \mathsf{Ok}(\bar{A}); \quad \langle x{:}T\rangle[\bar{A}] \triangleq (x{:}T, \mathsf{Ok}(\bar{A})); \quad \langle N\rangle[\bar{A}] \triangleq (x{:}\mathsf{Top}, \mathsf{Ok}(\bar{A}));$$
$$\langle x{:}T, nxts\rangle[\bar{A}] \triangleq (x{:}T, \langle nxts\rangle[\bar{A}]); \quad \langle N, nxts\rangle[\bar{A}] \triangleq (x{:}\mathsf{Top}, \langle nxts\rangle[\bar{A}])$$

Example 1: Establishing a session key using a nonce. Recall Example 1 from Section 4. Here are the types for the global names:

$$\begin{array}{l} \mathit{net} : \mathsf{Un} \qquad \bar{A}(n,k,p,q) \triangleq n\text{-}\mathsf{stamped}_{\mathsf{n}}(k{:}\mathsf{st\text{-}Key}(p,q), \mathsf{end}(\mathit{key}(p,k,q))) \\ \mathit{msg}_1 : \mathsf{Tag}(\,\langle n{:}\mathsf{Un}, k{:}\mathsf{Top}\rangle[\bar{A}(n,k,p,q)] \rightarrow \mathsf{lt\text{-}Auth}(l{:}\mathsf{lt\text{-}Key}(p,q), p{:}\mathsf{Un}, q{:}\mathsf{Un})\,) \\ \mathit{msg}_2 : \mathsf{Tag}(\,\langle m{:}\mathsf{st\text{-}Secret}\rangle[\mathsf{end}(\mathit{sec}(p,m,q))] \rightarrow \mathsf{st\text{-}Auth}(k{:}\mathsf{st\text{-}Key}(p,q), p{:}\mathsf{Un}, q{:}\mathsf{Un})\,) \end{array}$$

In the type of msg_1, note that the typing rules force us to stamp the type assertion $k{:}\mathsf{st\text{-}Key}(p,q)$. If we directly annotated the binder k by short-term type $\mathsf{st\text{-}Key}(p,q)$, then the protocol would not typecheck: Alice would not be permitted to form the message $\mathit{msg}_1\langle t, \mathit{kab}\rangle$ because $\mathit{step}(\mathsf{st\text{-}Key}(p,q)) \neq \mathsf{st\text{-}Key}(p,q)$ in violation to the premise for $\tau = \mathsf{lt}$ in the (Tag)-rule. Here is the type-annotated spi-calculus specification:

$$\begin{array}{l} P_A(a{:}\mathsf{Un}, b{:}\mathsf{Un}, \mathit{lab}{:}\mathsf{lt\text{-}Key}(a,b)) \triangleq \\ \quad \mathsf{inp}\ \mathit{net}\ (n{:}\mathsf{Un});\ \mathsf{new\ st\text{-}secret}\ (\mathit{kab}{:}\mathsf{st\text{-}Key}(a,b));\ \mathsf{begin!}(\mathit{key}(a,\mathit{kab},b)); \\ \quad \mathsf{new\ st\text{-}secret}\ (m{:}\mathsf{st\text{-}Secret});\ \mathsf{begin!}(\mathit{sec}(a,m,b)); \\ \quad \mathsf{out}\ \mathit{net}\ (\{\mathit{msg}_1\langle n, \mathit{kab}\rangle\}_{\mathit{lab}}, \{\mathit{msg}_2\langle m\rangle\}_{\mathit{kab}}) \\ P_B(a{:}\mathsf{Un}, b{:}\mathsf{Un}, \mathit{lab}{:}\mathsf{lt\text{-}Key}(a,b)) \triangleq \\ \quad \mathsf{new\ public}\ (n{:}\mathsf{Un});\ \mathsf{out}\ \mathit{net}\ n;\ \mathsf{inp}\ \mathit{net}\ (x{:}\mathsf{Un}, u{:}\mathsf{Un}); \\ \quad \mathsf{decrypt}\ x\ \mathsf{is}\ \{y{:}\mathsf{lt\text{-}Auth}(\mathit{lab}, a, b)\}_{\mathit{lab}}; \\ \quad \mathsf{match}\ z\ \mathsf{is}\ \mathit{msg}_1\langle n, \mathit{kab}{:}\mathsf{Top}\rangle[\bar{A}(n,\mathit{kab},a,b)];\ \mathsf{st\text{-}secret}(\mathit{kab});\ \mathsf{end}(\mathit{key}(a,\mathit{kab},b)); \\ \quad \mathsf{decrypt}\ u\ \mathsf{is}\ \{v{:}\mathsf{st\text{-}Auth}(\mathit{kab}, a, b)\}_{\mathit{kab}}; \\ \quad \mathsf{match}\ v\ \mathsf{is}\ \mathit{msg}_2\langle m{:}\mathsf{st\text{-}Secret}\rangle[\mathsf{end}(\mathit{sec}(a,m,b))]; \\ \quad \mathsf{st\text{-}secret}(m);\ \mathsf{end}(\mathit{sec}(a,m,b)) \end{array}$$

Example 2: Needham–Schroeder Symmetric Key Protocol (NSSK). This protocol is unsafe and, hence, does not typecheck. The problem is msg_3:

$$\begin{array}{l} \ldots \\ A \rightarrow B \qquad \{\mathit{msg}_3\langle A, \mathit{kab}\rangle\}_{\mathit{lbs}} \\ \ldots \end{array}$$

Here is the type that we want to give to the message tag:

$$\mathit{msg}_3 : \mathsf{Tag}(\,\langle p{:}\mathsf{Un}, k{:}\mathsf{st\text{-}Key}(p,q)\rangle[] \rightarrow \mathsf{lt\text{-}Auth}(l{:}\mathsf{lt\text{-}Key}(p,q), p{:}\mathsf{Un}, q{:}\mathsf{Un})\,)$$

However, this type does not permit Alice to form the tagged message $msg_3\langle A, kab\rangle$ because $step(\mathsf{st\text{-}Key}(p,q)) \neq \mathsf{st\text{-}Key}(p,q)$ in violation to the premises of (Tag).

Example 3: Denning–Sacco Protocol with acknowledgment. The Denning–Sacco protocol for establishing a short-term session key avoids the key compromise attack on NSSK by including a timestamp. We have added to the Denning–Sacco protocol Bob's acknowledgment for receipt of session key *kab*, which is achieved by Bob using *kab* to encrypt a tagged null-message.

$A \to S \qquad A, B$
S generates short-term secret kab and timestamp t
S begins! $init(S, kab, A, B)$ and $resp(S, kab, B, A)$
$S \to A \qquad \{msg_2\langle t, B, kab, \{msg_3\langle t, A, kab\rangle\}_{lbs}\rangle\}_{las}$
A asserts $\mathsf{st\text{-}secret}(kab)$ and ends $init(S, kab, A, B)$
$A \to B \qquad \{msg_3\langle t, A, kab\rangle\}_{lbs}$
B asserts $\mathsf{st\text{-}secret}(kab)$ and ends $resp(S, kab, B, A)$
B begins! $ack(B, kab, A)$
$B \to A \qquad \{msg_4\langle\rangle\}_{kab}$
A ends $ack(B, kab, A)$

The types for the long-term keys are las:$\mathsf{lt\text{-}Key}(A,S)$ and lbs:$\mathsf{lt\text{-}Key}(B,S)$. Here are the tag types:

$msg_2 : \mathsf{Tag}(\langle t{:}\mathsf{Un}, q{:}\mathsf{Un}, k{:}\mathsf{Top}, x{:}\mathsf{Un}\rangle[t\text{-}\mathsf{stamped_t}(k{:}\mathsf{st\text{-}Key}(p,q), \mathsf{end}(init(s,k,p,q)))]$
$\quad \to \mathsf{lt\text{-}Auth}(l{:}\mathsf{lt\text{-}Key}(p,s), p{:}\mathsf{Un}, s{:}\mathsf{Un}))$
$msg_3 : \mathsf{Tag}(\langle t{:}\mathsf{Un}, p{:}\mathsf{Un}, k{:}\mathsf{Top}\rangle[t\text{-}\mathsf{stamped_t}(k{:}\mathsf{st\text{-}Key}(p,q), \mathsf{end}(resp(s,k,q,p)))]$
$\quad \to \mathsf{lt\text{-}Auth}(l{:}\mathsf{lt\text{-}Key}(q,s), q{:}\mathsf{Un}, s{:}\mathsf{Un}))$
$msg_4 : \mathsf{Tag}(\langle\rangle[\mathsf{end}(ack(q,k,p))] \to \mathsf{st\text{-}Auth}(k{:}\mathsf{st\text{-}Key}(p,q), p{:}\mathsf{Un}, q{:}\mathsf{Un}))$

7 Type Preservation

Like in other type systems for spi-calculi, robust safety is a consequence of a type preservation theorem. In this section, we present this theorem and a few selected lemmas that are needed to prove it. Proofs and additional lemmas are omitted and given in an extended version of this paper. In order to define well-typed computation states, we extend the judgment for assertion entailment: Let $(E \vdash^+ A)$ iff it is derivable by the $\vdash$-rules plus the rules (Useless Nonce) and (Useless Time) below. The relation $\leq$ in (Useless Time) is defined by: $M \leq N$ iff either $M = N$ or $M = s \leq t = N$ for times $s, t \in \mathbb{N}$.

Well-typed Computation States, $(t; \vec{n}; \mathsf{end}\{\vec{M}\} \mid\mid P) : \diamond$**:**

$$\frac{\text{(Good State)} \quad E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t)) \quad E \vdash^+ \overline{A} \quad \overline{A} \vdash P \quad \vec{n} \text{ distinct} \quad \vec{T} \text{ generative}}{(t; \vec{n}; \mathsf{end}\{\vec{M}\} \mid\mid P) : \diamond}$$

$$\frac{\text{(Useless Nonce)} \quad fv(A) \subseteq fv(E) \quad \mathsf{fresh}(N) \notin E \quad E \vdash^+ N : \mathsf{Top}}{E \vdash^+ N\text{-}\mathsf{stamped_n}(A)} \qquad \frac{\text{(Useless Time)} \quad fv(A) \subseteq fv(E) \quad (\forall M)(\mathsf{now}(M) \in E \Rightarrow M \not\leq N) \quad E \vdash^+ N : \mathsf{Top}}{E \vdash^+ N\text{-}\mathsf{stamped_t}(A)}$$

The additional rules (Useless Nonce) and (Useless Time) allow to stamp assertions with messages that are neither fresh nonces nor current or future times. This is safe because the typing rules for unstamping are not applicable in such cases. Technically, these rules are needed to prove the following lemma.

Lemma (Step Invariance) If E is basic and $(E \vdash^+ \bar{A})$, then $(\mathit{step}^+(E) \vdash^+ \mathit{step}(\bar{A}))$.

Proof $\mathit{step}^+/\mathit{step}$ maps (Nonce Stamp) to (Useless Nonce), and (Time Stamp) to (Useless Time). (Encrypt) is mapped to itself if $\tau = \mathsf{lt}$ and to (Encrypt Un) if $\tau = \mathsf{st}$. (Tag) is mapped to itself if $\tau = \mathsf{lt}$ and to (Tag Un) if $\tau = \mathsf{st}$. □

Here are the definitions that are needed to fully understand the step invariance lemma: We call environment E *basic* iff it is of the form $E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t))$ for distinct $\vec{n}$ and generative $\vec{T}$. Let $\mathit{step}^+(E, \mathsf{now}(t)) \stackrel{\Delta}{=} (\mathit{step}(E), \mathsf{now}(t+1))$.

Lemma (Cut) *If E is basic,* $(E \vdash^+ \bar{A})$ *and* $(\bar{A} \vdash \bar{B})$*, then* $(E \vdash^+ \bar{B})$.

This cut lemma is not hard to prove. Step invariance and cut are used to prove that tick-reductions preserve well-typedness. More generally, we obtain the following theorem.

Theorem (Type Preservation) *If* $((t;\vec{n};\bar{A} \mid\mid P) : \diamond)$ *and* $(t;\vec{n};\bar{A} \mid\mid P) \Rightarrow (s;\vec{m};\bar{B} \mid\mid Q)$*, then* $((s;\vec{m};\bar{B} \mid\mid Q) : \diamond)$.

8 Conclusion

Related Work. Compared to other work on the spi-calculus [3, 1, 11, 12, 10, 2, 16], the novelty of this paper is the addition of time, key-compromising attackers and short-term assertions for secrecy and authenticity. To the best of our knowledge, this is the first spi-calculus type system for reasoning about short-term assertions.

There are some formal models for cryptographic protocols that deal with recency or key compromise implicitly (without explicitly modeling time): BAN logic [5] has a primitive formula for freshness, which allows reasoning about recency. Guttman shows how to reason about recency for nonce-based protocols within the strand space model [15]. Both these works are based on the assumption that protocol sessions time out before short-term keys can possibly get compromised. Paulson's inductive method [21] models key compromise by a rule called "Oops" for leaking short-term keys, and his safety theorems typically require premises that certain data has not been leaked to dishonest principals. Recently, Gordon and Jeffrey [13] have presented a type system for proving conditional secrecy that models key compromise in a similar way.

There are also some models that deal with time more explicitly: Evans and Schneider [9] analyze time dependent security properties in tock-CSP using theorem proving with the rank function method. Rank functions have similarities with type systems: on the one hand, both rank functions and type systems are designed to prove safety properties without assuming a bounded number of sessions and, on the other hand, both require less help from protocol specifiers than general theorem proving—the supply of a rank-function or type-annotations is enough. Gorrieri, Locatelli and Martinelli [14] present the process algebra tCryptoSPA with event-based time for expressing cryptographic protocols. Both tock-CSP and tCryptoSPA seem a bit more expressive than our timed spi-calculus. For instance, these languages can express processes that patiently

wait for input arbitrarily long, whereas it is not obvious how to express this in our language. On the other hand, both tock-CSP and tCryptoSPA allow some anomalies, like timestops, that our language omits. Bozga, Ene and Lakhnech [19] and Delzanno and Ganty [7] model real time and present symbolic procedures for checking time sensitive safety properties. These procedures are more automatic than typechecking; they do not require help in the form of type annotations. On the other hand, they assume a bounded number of sessions [19] or do not guarantee termination [7]. The Casper model checker is based on discretely timed CSP and can analyze protocols for timed agreement and timed secrecy [22]. It requires bounds on the size of protocols. Most of the languages discussed in this paragraph, with the exception of [19], do not explicitly model key compromising attackers. [19] models key compromising attackers by creating for each short-term key a key-cracking process that accepts messages encrypted under this key, then waits for a while and then publishes the key. In this model, attackers can crack keys by directing ciphertexts to key-cracking processes. Key compromising attackers could probably be modeled similarly in the other languages above, but we expect that key-cracking processes create additional problems for some of the verification methods.

Limitations of our Model of Time. A limitation is that we cannot distinguish between the amounts of time that it takes to timeout and the time needed for cracking short-term keys. In reality, the former is usually much shorter than the latter. While it is often safe to assume that timeout happens later than it really does, it sometimes prevents us from expressing attacks. Consider, for instance, the Wide Mouthed Frog protocol (WMF):

$$
\begin{array}{l}
A \text{ generates short-term key } kab \\
A \to S \qquad A, \{t_i, B, kab\}_{kas} \\
S \to B \qquad \{t_s, A, kab\}_{kbs} \\
B \text{ asserts } \mathsf{st\text{-}secret}(kab)
\end{array}
$$

There is a type confusion attack on WMF, where the attacker repeatedly intercepts the second message and plays it back to the server as the first message of another run. The attacker, thus, always has a message of the correct format that contains a current timestamp and after cracking *kab* he can fool either principal to accept the compromised *kab* as recent. In our model, the server's timestamp t_s will always be equal to the initiator's timestamp t_i. Therefore, the attack is not possible in the model. (Fortunately, WMF still does not typecheck, though.) While it would not be hard to refine our model of time, it is less clear how to refine the type system.

Future Work. Although this article deals with symmetric cryptography only, we expect no problems to integrate this work into a more general system with public cryptography and other cryptographic operators [12, 16]. Our type system is simple and can typecheck many key distribution protocols from the literature [6]. While we plan to investigate how it can be extended to verify protocols with additional intricacies, like Yahalom [5], we do not think that such extensions are of utmost importance, because often similar, sometimes simpler, protocols exist that achieve the same security goals and obey our type discipline, for instance BAN's Yahalom simplification [5]. More interestingly, we plan to investigate if we can find similar type systems for refined models of time.

References

1. M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, September 1999.
2. M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *Foundations of Software Science and Computation Structures*, volume 2030 of *LNCS*. Springer, 2001.
3. M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999.
4. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.
5. M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989.
6. J. Clark and J. Jacob. A survey of authentication protocol literature. Unpublished report. University of York, 1997.
7. G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *LNCS*, pages 342–356. Springer, 2004.
8. D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.
9. N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *ESORICS*, volume 1895 of *LNCS*, pages 222–237. Springer, 2000.
10. A. D. Gordon and A.S.A. Jeffrey. Typing one-to-one and one-to-many correspondences in security protocols. In *Proc. Int. Software Security Symp.*, volume 2609 of *Lecture Notes in Computer Science*, pages 263–282. Springer-Verlag, 2002.
11. A.D. Gordon and A.S.A. Jeffrey. Authenticity by typing for security protocols. *J. Computer Security*, 11(4):451–521, 2003.
12. A.D. Gordon and A.S.A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *J. Computer Security*, 12(3/4):435–484, 2003.
13. A.D. Gordon and A.S.A. Jeffrey. Secrecy despite compromise: Types, cryptography and the pi-calculus. In *CONCUR 2005: Concurrency Theory*, LNCS. Springer, 2005.
14. R. Gorrieri, E. Locatelli, and F. Martinelli. A simple language for realtime cryptographic protocol analysis. In P. Degano, editor, *12th European Symposium on Programming*, volume 2618 of *LNCS*, pages 114–128. Springer, 2003.
15. Joshua D. Guttman. Key compromise, strand spaces, and the authentication tests. *Electr. Notes Theor. Comput. Sci.*, 45, 2001.
16. C. Haack and A.S.A. Jeffrey. Pattern-matching spi-calculus. In *2nd IFIP Workshop on Formal Aspects in Security and Trust*, volume 173 of *IFIP*. Kluwer Academic Press, 2004.
17. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *13th IEEE Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society Press, 2000.
18. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.
19. Y. Lakhnech L. Bozga, C. Ene. A symbolic decision procedure for cryptographic protocols with time stamps. In *CONCUR 2004: Concurrency Theory*, volume 3170 of *LNCS*, pages 177–192. Springer, 2004.
20. R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
21. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
22. P. Ryan and S. Schneider. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2001.

Proofs

Lemma 1 (Transitivity of Subtyping). *If $T \leq U$ and $U \leq V$, then $T \leq V$.*

Proof By induction on the derivation of $T \leq U$. Suppose $\mathcal{D}_1 \rhd (T \leq U)$ and $\mathcal{D}_2 \rhd (U \leq V)$. If the last rule of either $\mathcal{D}_1$ or $\mathcal{D}_2$ is either (Sub Refl) or (Sub Top), we easily get that $T \leq V$. So suppose that neither $\mathcal{D}_1$ nor $\mathcal{D}_2$ ends in either of these rules. An inspection of the other subtyping rules shows that then either both $\mathcal{D}_1$ and $\mathcal{D}_2$ end in (Sub Pair), or $\mathcal{D}_1$ ends in (Sub Pair) and $\mathcal{D}_2$ in (Sub Pair Un). In the former case, we get $T \leq V$ by induction hypothesis and (Sub Pair). In the latter case, we get $T \leq V$ by induction hypothesis and (Sub Pair Un). □

The following facts are all quite obvious.

Lemma 2 (Basic Facts). *$fv(step(T)) \subseteq fv(T)$; $fv(step(\bar{A})) \subseteq fv(\bar{A})$; $subj(step(E)) = subj(E)$; $step(T)\{\rho\} = step(T\{\rho\})$; $step(A)\{\rho\} = step(A\{\rho\})$; $step(step(E)) = step(E)$; $subj(E)\{\rho\} = subj(E\{\rho\})$; $T \leq U \Leftrightarrow T\{\rho\} \leq U\{\rho\}$.*

Lemma 3 (Free Variables). *If $(E \vdash \mathcal{R})$, then $(E \vdash \diamond)$ and $fv(\mathcal{R}) \subseteq fv(E)$.*

Proof By induction on $(E \vdash \mathcal{R})$'s derivation. □

Lemma 4 (Environment Union). *If $(E \vdash \diamond)$ and $(F \vdash \diamond)$, then $(E,F \vdash \diamond)$.*

Proof Recall that, by definition, $(E \vdash \diamond)$ iff $fv(E) \subseteq fv(subj(E))$. We have $fv(E,F) = fv(E) \cup fv(F) \subseteq fv(subj(E)) \cup fv(subj(F)) = fv(subj(E) \cup subj(F)) = fv(subj(E,F))$. □

Lemma 5 (Weakening). *If $(E,\bar{A} \vdash \diamond)$ and $(E \vdash \mathcal{R})$, then $(E,\bar{A} \vdash \mathcal{R})$.*

Proof By induction on $(E \vdash \mathcal{R})$'s derivation. □

Lemma 6 (Specialization). *If $(E,M{:}U \vdash \mathcal{R})$ and $(E \vdash T \leq U)$, then $(E,M{:}T \vdash \mathcal{R})$.*

Proof By induction on $(E,M{:}U \vdash \mathcal{R})$'s derivation. □

Lemma 7 (Substitutivity). *If $(E \vdash \mathcal{R})$, then $(E\{\rho\} \vdash \mathcal{R}\{\rho\})$.*

Proof By induction on $(E \vdash \mathcal{R})$'s derivation. Suppose $(E \vdash \mathcal{R})$ and let ρ be a substitution.

Proof case, $\mathcal{R} = \diamond$: We know that $fv(E) \subseteq fv(subj(E))$ and want to show that $fv(E\{\rho\}) \subseteq fv(subj(E\{\rho\}))$. Let $x \in fv(E\{\rho\})$. We distinguish two subcases. Suppose first that $x \in fv(E) - dom(\rho)$. Then $x \in fv(subj(E)) - dom(\rho)$. Then $x \in fv(subj(E)\{\rho\}) = fv(subj(E\{\rho\}))$. Suppose now that $x \in fv(\rho(y))$ and $y \in fv(E) \cap dom(\rho)$. Then $y \in fv(subj(E))$. Then $x \in fv(\rho(y)) \subseteq fv(subj(E)\{\rho\}) = fv(subj(E\{\rho\}))$.

Proof case, (Tag): We know:

- $\rho' = (k, \vec{x} \leftarrow K, \vec{N})$
- $\tau = \mathsf{st} \Rightarrow step(T) \leq \mathsf{Un}$
- $\tau = \mathsf{lt} \Rightarrow step(T,U,\vec{V}) = (T,U,\vec{V})$
- $E \vdash L : \mathsf{Tag}(T \rightarrow \tau\text{-}\mathsf{Auth}(k{:}U, \vec{x}{:}\vec{V})), M{:}T\{\rho'\}, K{:}U\{\rho'\}, \vec{N}{:}\vec{V}\{\rho'\}$

By renaming of bound variables, we may assume that $(dom(\rho) \cup ran(\rho)) \cap \{k, \vec{x}\} = \emptyset$. Note that applying the substitution ρ preserves the two implications for $\tau = \mathsf{st}$ and $\tau = \mathsf{lt}$. Define $\rho'' \triangleq (k, \vec{x} \leftarrow K\{\rho\}, \vec{N}\{\rho\})$. Note that $(\rho'; \rho) = (\rho; \rho'')$. By induction hypothesis, we get:

- $E\{\rho\} \vdash L\{\rho\} : \mathsf{Tag}(T\{\rho\} \to \tau\text{-}\mathsf{Auth}(k{:}U\{\rho\}, \vec{x}{:}\vec{V}\{\rho\}))$,
  $M\{\rho\} : T\{\rho\}\{\rho''\}, K\{\rho\} : U\{\rho\}\{\rho''\}, \vec{N}\{\rho\} : \vec{V}\{\rho\}\{\rho''\}$

Then by (Tag):

- $E\{\rho\} \vdash L\{\rho\}(M\{\rho\}) : \tau\text{-}\mathsf{Auth}(K\{\rho\}, \vec{N}\{\rho\})$

Proof case, (Untag): We know:

- $\rho' = (k, \vec{y} \leftarrow K, \vec{N})$
- $E \vdash T\{\rho'\} \leq U$
- $E \vdash M : \tau\text{-}\mathsf{Auth}(K, \vec{N})$, $L : \mathsf{Tag}(T \to \tau\text{-}\mathsf{Auth}(k{:}U, \vec{y}{:}\vec{V}))$
- $E, x : U \vdash P$

By renaming of bound variables, we may assume that $(dom(\rho) \cup ran(\rho)) \cap \{x, k, \vec{y}\} = \emptyset$. Define $\rho'' \triangleq (k, \vec{y} \leftarrow K\{\rho\}, \vec{N}\{\rho\})$. Note that $(\rho'; \rho) = (\rho; \rho'')$. By induction hypothesis, we get:

- $E\{\rho\} \vdash T\{\rho\}\{\rho''\} \leq U\{\rho\}$
- $E\{\rho\} \vdash M\{\rho\} : \tau\text{-}\mathsf{Auth}(K\{\rho\}, \vec{N}\{\rho\}), L\{\rho\} : \mathsf{Tag}(T\{\rho\} \to \tau\text{-}\mathsf{Auth}(k{:}U\{\rho\}, \vec{y}{:}\vec{V}\{\rho\}))$
- $E\{\rho\}, x : U\{\rho\} \vdash P\{\rho\}$

Then by (Untag):

- $E\{\rho\} \vdash \mathsf{untag}\ M\{\rho\}\ \mathsf{is}\ L\{\rho\}(x{:}U\{\rho\}); P\{\rho\}$

The other proof cases are similar. □

Let $M \leq N$ iff either $M = N$ or $M = s \leq t = N$ for times $s, t \in \mathbb{N}$. Let $(E \vdash^+ \overline{A})$ iff it is derivable by the $\vdash$-rules plus the rules (Useless Nonce) and (Useless Time) below.

Good Computation States, $(t; \vec{n}; \mathsf{end}\{\vec{M}\} \mid\mid P) : \diamond$**:**

$$\frac{\text{(Good State)} \quad E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t)) \quad E \vdash^+ \overline{A} \quad \overline{A} \vdash P \quad \vec{n}\ \text{distinct} \quad \vec{T}\ \text{generative}}{(t; \vec{n}; \mathsf{end}\{\vec{M}\} \mid\mid P) : \diamond}$$

$$\frac{\text{(Useless Nonce)} \quad fv(A) \subseteq fv(E) \quad \mathsf{fresh}(N) \notin E \quad E \vdash^+ N : \mathsf{Top}}{E \vdash^+ N\text{-}\mathsf{stamped}_{\mathsf{n}}(A)} \qquad \frac{\text{(Useless Time)} \quad fv(A) \subseteq fv(E) \quad (\forall M)(\mathsf{now}(M) \in E \Rightarrow M \not\leq N) \quad E \vdash^+ N : \mathsf{Top}}{E \vdash^+ N\text{-}\mathsf{stamped}_{\mathsf{t}}(A)}$$

Lemma 8 (Free Variables). *If* $(E \vdash^+ \mathcal{R})$*, then* $(E \vdash \diamond)$ *and* $fv(\mathcal{R}) \subseteq fv(E)$.

Proof Like the proof of Lemma 3 with two additional proof cases for the additional rules. □

$\vdash^+$ does not satisfy weakening. For instance, $(n : \mathsf{Top} \vdash^+ n\text{-}\mathsf{stamped}_{\mathsf{n}}(\mathsf{end}()))$ but $(n : \mathsf{Top}, \mathsf{fresh}(n) \not\vdash^+ n\text{-}\mathsf{stamped}_{\mathsf{n}}(\mathsf{end}()))$. However, the following instances of weakening hold:

Lemma 9 (Weakening). *If* $(E, \mathsf{end}(M) \vdash \diamond)$ *and* $(E \vdash^+ \bar{A})$, *then* $(E, \mathsf{end}(M) \vdash^+ \bar{A})$. *If* $n \notin fv(E)$, $(E, n{:}T, \mathsf{fresh}(n) \vdash \diamond)$ *and* $(E \vdash^+ \bar{A})$, *then* $(E, n{:}T, \mathsf{fresh}(n) \vdash^+ \bar{A})$.

Proof Each of the two statements is proved by induction of the derivation, just like in the proof of Lemma 5. The only rules that may possibly cause problems are the new rules (Useless Nonce) and (Useless Time). It is easy to see that neither of these rules causes problems for the first statement. It is also easy to see that (Useless Time) does not cause a problem for the second statement. So the only rule that is potentially problematic is (Useless Nonce) in the proof of the second statement. We will do this proof case: Suppose that $n \notin fv(E)$, $(E, n{:}T, \mathsf{fresh}(n) \vdash \diamond)$ and $(E \vdash^+ n\text{-}\mathsf{stamped}_{\mathsf{n}}(A))$. Suppose, furthermore, that the derivation of this last judgment ends in (Useless Nonce). Then $(E \vdash^+ n : \mathsf{Top})$. Then $n \in fv(E)$, by Lemma 8. But that contradicts our assumption that $n \notin fv(E)$. □

Call environment E *basic* iff it is of the form $E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t))$ for distinct $\vec{n}$ and generative $\vec{T}$. Let $step^+(E, \mathsf{now}(t)) \triangleq (step(E), \mathsf{now}(t+1))$.

Lemma 10 (Step Invariance). *Let* E *be basic. If* $(E \vdash \diamond)$, *then* $(step^+(E) \vdash \diamond)$. *If* $(E \vdash T \leq U)$, *then* $(step^+(E) \vdash step(T) \leq step(U))$. *If* $(E \vdash^+ \bar{A})$, *then* $(step^+(E) \vdash^+ step(\bar{A}))$.

Proof By induction on the derivation of $(E \vdash^+ \mathcal{R})$. $step^+$/$step$ maps (Nonce Stamp) to (Useless Nonce), and (Time Stamp) to (Useless Time). (Useless Nonce) is mapped to itself and so is (Useless Time). (Encrypt) is mapped to itself if $\tau = \mathsf{lt}$ and to (Encrypt Un) if $\tau = \mathsf{st}$. (Tag) is mapped to itself if $\tau = \mathsf{lt}$ and to (Tag Un) if $\tau = \mathsf{st}$. □

Lemma 11 (Cut). *If* E *is basic,* $(E \vdash^+ \bar{A})$ *and* $(\bar{A} \vdash \bar{B})$, *then* $(E \vdash^+ \bar{B})$.

Proof By induction on $(\bar{A} \vdash \bar{B})$'s derivation. □

Lemma 12 (Name Types). *If* $(E \vdash^+ n{:}U)$, *then* $E = (E', n{:}T)$ *and* $(E \vdash T \leq U)$ *for some* T, E'.

Proof The type U of n can only have been introduced by (Id), possibly followed by one or more (Sub). □

Lemma 13 (Trusted Keys and Tags are Names). *Suppose* E *is basic.*

(a) *If* $(E \vdash^+ K : \tau\text{-}\mathsf{Key}(\vec{N}))$, *then* K *is a name and* $E = (E', K : \tau\text{-}\mathsf{Key}(\vec{N}))$ *for some* E'.
(b) *If* $(E \vdash^+ L : \mathsf{Tag}(X))$, *then* L *is a name and* $E = (E', L : \mathsf{Tag}(X))$ *for some* E'.

Proof The rules for encryption and tagging do not introduce key or tag types. Subsumption does not either, because key and tag types are minimal types. Therefore, key and tag types only get introduced by (Id). □

Lemma 14. *If* E *is basic, then* $(E \not\vdash^+ M : \tau\text{-}\mathsf{Secret}, M : \mathsf{Un})$.

Proof Suppose E is basic and, towards a contradiction, $(E \vdash^+ M : \tau\text{-}\mathsf{Secret}, M : \mathsf{Un})$. If M was a variable, then Un and $\tau\text{-}\mathsf{Secret}$ would have a common subtype, by Lemma 12 and functionality of basic environments. But Un and $\tau\text{-}\mathsf{Secret}$ do not have a common subtype, and so M is not a variable. M cannot be of the form $M = \{N\}_K$, because

τ-Secret is not a supertype of Un. M cannot be of the form $M = L(N)$, because τ-Secret is neither a supertype of Un nor of τ'-Auth$(K, \vec{N}')$ for any $\tau', K, \vec{N}'$. M cannot be a pair, because τ-Secret is neither a supertype of a pair type nor Un. M cannot be the empty message, because τ-Secret is not a supertype of any ok-type. □

The following lemma is essential for proving type preservation for the reduction semantics. Its proof is less obvious than in type systems that are purely syntax-directed. The difficulty arises because of subtyping and rules like (Encrypt Un) and (Tag Un), which are necessary to model attackers as well-typed processes. This is typical for type systems for cryptographic protocols. The type system is carefully designed so that the following inversion properties hold.

Lemma 15 (Rule Inversions). *Suppose E is basic.*

(a) *If* $(E \vdash^+ \{M\}_K : \mathsf{Un}, K : \tau\text{-Key}(\vec{N}))$, *then* $(E \vdash^+ M : \tau\text{-Auth}(K, \vec{N}))$.
(b) *If* $(E \vdash^+ \{M\}_K : \mathsf{Un}, K : \mathsf{Un})$, *then* $(E \vdash^+ M : \mathsf{Un})$.
(c) *If* $(E \vdash^+ L(M) : \tau\text{-Auth}(K, \vec{N}), L : \mathsf{Tag}(T \to \tau\text{-Auth}(k{:}U, \vec{x}{:}\vec{V})))$,
then $(E \vdash^+ M : T\{k, \vec{x} \leftarrow K, \vec{N}\})$.
(d) *If* $(E \vdash^+ L(M) : \mathsf{Un})$, *then* $(E \vdash^+ M : \mathsf{Un})$.
(e) *If* $(E \vdash^+ (M, N) : (x{:}T, U))$, *then* $(E \vdash^+ M : T, N : U\{x \leftarrow M\})$.
(f) *If* $(E \vdash^+ (M, N) : \mathsf{Un})$, *then* $(E \vdash^+ M : \mathsf{Un}, N : \mathsf{Un})$.
(g) *If* $(E \vdash^+ M : \mathsf{Ok}(A))$, *then* $(E \vdash^+ A)$.
(h) *If* $(E \vdash^+ \mathsf{fresh}(N), N\text{-stamped}_{\mathsf{n}}(A))$, *then* $(E \vdash^+ A)$.
(i) *If* $(E \vdash^+ \mathsf{now}(N), N\text{-stamped}_{\mathsf{t}}(A))$, *then* $(E \vdash^+ A)$.

Proof Suppose E is basic.

Part (a)*:* Suppose $(E \vdash^+ \{M\}_K : \mathsf{Un}, K : \tau\text{-Key}(\vec{N}))$. The derivation of this judgment can only end in (And), and we obtain $(E \vdash^+ \{M\}_K : \mathsf{Un})$ and $(E \vdash^+ K : \tau\text{-Key}(\vec{N}))$. We strip off all pointless uses of (Sub)/(Sub Refl) from the end of $(E \vdash^+ \{M\}_K : \mathsf{Un})$'s derivation. If the resulting derivation ended in (Encrypt Un), then we would have $(E \vdash^+ K : \tau\text{-Key}(\vec{N}), K : \mathsf{Un})$, in contradiction to Lemma 14. Moreover, the derivation cannot end in (Id), because E is basic. Therefore, the derivation can only end in (Encrypt). Then $(E \vdash^+ M : \tau\text{-Auth}(K, \vec{N}))$.

Part (b)*:* Similar to proof of part (a).

Part (c)*:* Let $X = (T \to \tau\text{-Auth}(k{:}U, \vec{x}{:}\vec{V}))$ and $(E \vdash L(M) : \tau\text{-Auth}(K, \vec{N}), L : \mathsf{Tag}(X))$. The derivation of this judgment can only end in (And). So $(E \vdash L(M) : \tau\text{-Auth}(K, \vec{N}))$ and $(E \vdash L : \mathsf{Tag}(X))$. The former judgment's derivation can only end in either (Tag) or (Tag Un) followed by a possibly empty sequence of applications of (Sub). So there are $\mathcal{D}$, W such that $\mathcal{D} \rhd (E \vdash L(M) : W)$, $(W \leq \tau\text{-Auth}(K, \vec{N}))$ and $\mathcal{D}$ ends in (Tag Un) or (Tag). Because $\mathsf{Un} \not\leq \tau\text{-Auth}(K, \vec{N})$, $\mathcal{D}$ does not end in (Tag Un), so it ends in (Tag). Because $\tau\text{-Auth}(K, \vec{N})$ is a minimal type, $W = \tau\text{-Auth}(K, \vec{N})$. So there is a type scheme $X' = (T' \to \tau\text{-Auth}(k{:}U', \vec{x}{:}\vec{V}'))$ such that $(E \vdash^+ L : \mathsf{Tag}(X'))$ and $(E \vdash^+ M : T'\{k, \vec{x} \leftarrow K, \vec{N}\})$. By Lemmas 13 and 12 and functionality of E, it follows that $\mathsf{Tag}(X)$ and $\mathsf{Tag}(X')$ have a common subtype. Because tag types are minimal types, it follows that $X = X'$. In particular, $T = T'$. Thus, $(E \vdash^+ M : T\{k, \vec{x} \leftarrow K, \vec{N}\})$.

Part (d)*:* Suppose $(E \vdash L(M) : \mathsf{Un})$. Then there are T, $\mathcal{D}$ such such that $\mathcal{D} \rhd (E \vdash L(M) : T)$, $(T \leq \mathsf{Un})$ and $\mathcal{D}$ ends in either (Tag) or (Tag Un). Because authentication

types are not subtypes of Un, $\mathcal{D}$ cannot end in (Tag), so it ends in (Tag Un). Then $(E \vdash M : \mathsf{Un})$.

Part (e)*:* Suppose $(E \vdash (M,N) : (x{:}T,U))$. Then there are $T',U',\mathcal{D}$ such that $\mathcal{D} \rhd (E \vdash (M,N) : (x{:}T',U'))$, $(x{:}T',U') \leq (x{:}T,U)$ and $\mathcal{D}$ ends in (Pair). From $(x{:}T',U') \leq (x{:}T,U)$ it follows that $T' \leq T$ and $U' \leq U$, by inspection of the subtyping rules. By substitutivity of subtyping (Lemma 2), we get $U'\{x{\leftarrow}M\} \leq U\{x{\leftarrow}M\}$. Because $\mathcal{D}$ ends in (Pair), we have $(E \vdash M : T', N : U'\{x{\leftarrow}M\})$. Then $(E \vdash M : T, N : U\{x{\leftarrow}M\})$, by (Sub).

Part (f)*:* Similar to proof of part (f).

Part (g)*:* Suppose $(E \vdash M : \mathsf{Ok}(\bar{A}))$. Then there are T, $\mathcal{D}$ such that $\mathcal{D} \rhd (E \vdash M : T)$, $(T \leq \mathsf{Ok}(\bar{A}))$ and $\mathcal{D}$ ends in either (Id) or (Empty). If $\bar{A} = \emptyset$, then trivially $(E \vdash \bar{A})$ by (And). So suppose that $\bar{A} \neq \emptyset$. Then $\mathsf{Ok}(\bar{A})$ is not generative, so $\mathcal{D}$ does not end in (Id), so it ends in (Empty). Then $(E \vdash \bar{A})$.

Part (h)*:* Suppose $(E \vdash^{+} \mathsf{fresh}(N), N\text{-}\mathsf{stamped}_{\mathsf{n}}(A))$. The derivation of this judgment can only end in (And), and we obtain $(E \vdash^{+} \mathsf{fresh}(N))$ and $(E \vdash N\text{-}\mathsf{stamped}_{\mathsf{n}}(A))$. The derivation of the former can only end in (Id), so $\mathsf{fresh}(N) \in E$. Then the derivation of the latter cannot end in (Useless Nonce), so it ends in (Nonce Stamp). Then $(E \vdash A)$.

Part (i)*:* Suppose $(E \vdash^{+} \mathsf{now}(N), N\text{-}\mathsf{stamped}_{\mathsf{t}}(A))$. The derivation of this judgment can only end in (And), and we obtain $(E \vdash^{+} \mathsf{now}(N))$ and $(E \vdash N\text{-}\mathsf{stamped}_{\mathsf{n}}(A))$. The derivation of the former can only end in (Id), so $\mathsf{now}(N) \in E$. Then the derivation of the latter cannot end in (Useless Time), so it ends in (Time Stamp). Then $(E \vdash A)$. □

We have to do a bit of proof normalization to cope with the fact that the process-level rules (Nonce Unstamp), (Time Unstamp) and (Ok) are not syntax-directed. Call these three rule *critical*. Call a derivation $\mathcal{D} \rhd (E \vdash P)$ *normal* iff it has no subderivations $\mathcal{D}'$ of the form $\mathcal{D}' = ((\mathcal{D}'_1, \mathcal{D}'_2), (E \vdash Q \mid R), (\text{Par}))$ where $\mathcal{D}'_1$ or $\mathcal{D}'_2$ ends in a critical rule, or $\mathcal{D}' = ((\mathcal{D}'_1, \mathcal{D}'_2), (E \vdash !Q), (\text{Repl}))$ where $\mathcal{D}'_1$ ends in a critical rule. Call a derivation $\mathcal{D} = ((\mathcal{D}_1, \mathcal{D}_2), ((t;\vec{n};\bar{A} \mid\mid P) : \diamond), (\text{Good State}))$ *normal* iff $\mathcal{D}_2$ is normal and does not end in a critical rule.

Lemma 16 (Normal Derivations). *Every derivable judgment has a normal derivation.*

Proof If a (Par)-node has critical rule as a child, they can be permuted. For instance:

$$\dfrac{\dfrac{E \vdash M : \mathsf{Ok}(\bar{A}) \qquad E,\bar{A} \vdash P}{E \vdash P}\,(\text{Ok}) \qquad E \vdash Q}{E \vdash P \mid Q}\,(\text{Par})$$

$$\Downarrow$$

$$\dfrac{E \vdash M : \mathsf{Ok}(\bar{A}) \qquad \dfrac{E,\bar{A} \vdash P \qquad E,\bar{A} \vdash Q}{E,\bar{A} \vdash P \mid Q}\,(\text{Par})}{E \vdash P \mid Q}\,(\text{Ok})$$

Similarly if the first child of (Repl) is a critical rule:

$$\dfrac{\dfrac{E \vdash M : \mathsf{Ok}(\bar{A}) \qquad E, \bar{A} \vdash P}{E \vdash P}\,(\mathrm{Ok}) \qquad step(E) \vdash P}{E \vdash !P}\,(\mathrm{Repl})$$

$$\Downarrow$$

$$\dfrac{E \vdash M : \mathsf{Ok}(\bar{A}) \qquad \dfrac{E, \bar{A} \vdash P \qquad step(E, \bar{A}) \vdash P}{E, \bar{A} \vdash P}\,(\mathrm{Repl})}{E \vdash !P}\,(\mathrm{Ok})$$

If the second child of (Good State) is a critical rule, this can be eliminated using cut and Lemma 15(g), (h), (i). For instance, suppose that a derivation has the following shape:

$$\dfrac{\mathsf{now}(t), \vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\} \vdash^{+} \bar{A} \qquad \dfrac{\bar{A} \vdash M : \mathsf{Ok}(\bar{B}) \qquad \bar{A}, \bar{B} \vdash P}{\bar{A} \vdash P}\,(\mathrm{Ok})}{(t; \vec{n}; \mathsf{end}\{\vec{M}\} \parallel P) : \diamond}\,(\text{Good State})$$

Let $E = (\mathsf{now}(t), \vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\})$. From $(E \vdash^{+} \bar{A})$ and $(\bar{A} \vdash M : \mathsf{Ok}(\bar{B}))$ we obtain $(E \vdash M : \mathsf{Ok}(\bar{B}))$, by cut. Then $(E \vdash^{+} \bar{B})$, by Lemma 15(g). Then $(E \vdash^{+} \bar{A}, \bar{B})$, by (And). So we now have a derivation of the following form:

$$\dfrac{\mathsf{now}(t), \vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\} \vdash^{+} \bar{A}, \bar{B} \qquad \bar{A}, \bar{B} \vdash P}{(t; \vec{n}; \mathsf{end}\{\vec{M}\} \parallel P) : \diamond}\,(\text{Good State})$$

In the same way, we can eliminate all remaining critical rules from the end of $\mathcal{D}_3$. □

Call a derivation *subnormal* iff it is normal and does not end in a critical rule.

Lemma 17 (Congruence Preserves Typing). *If $P \equiv Q$, then $(E \vdash P)$ has a subnormal derivation iff $(E \vdash Q)$ has a subnormal derivation.*

Proof By induction on $(P \equiv Q)$'s derivation. □

Lemma 18 (Instantaneous Reduction Preserves Typing). *If $((t; \vec{n}; \bar{A} \parallel P) : \diamond)$ and $(t; \vec{n}; \bar{A} \parallel P) \rightarrow (t; \vec{m}; \bar{B} \parallel Q)$, then $((t; \vec{m}; \bar{B} \parallel Q) : \diamond)$.*

Proof By induction on $(t; \vec{n}; \bar{A} \parallel P) \rightarrow (t; \vec{m}; \bar{B} \parallel Q)$'s derivation. Let $\mathcal{D} \rhd ((t; \vec{n}; \bar{A} \parallel P) : \diamond)$ be normal and $(t; \vec{n}; \bar{A} \parallel P) \rightarrow (t; \vec{m}; \bar{B} \parallel Q)$. From the premises of $\mathcal{D}$'s last rule, we obtain a basic environment $E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t))$, where $\bar{A} = \mathsf{end}\{\vec{M}\}$, and $\bar{C}, \mathcal{D}'$ such that $(E \vdash^{+} \bar{C})$ and $\mathcal{D}' \rhd (C \vdash P)$ is subnormal.

Proof case, (Redn Equiv): In this case, $P \equiv P'$, $(t; \vec{n}; \bar{A} \parallel P') \rightarrow (t; \vec{m}; \bar{B} \parallel Q')$ and $Q' \equiv Q$. By Lemma 17, $(\bar{C} \vdash P')$, thus $((t; \vec{n}; \bar{A} \parallel P') : \diamond)$. By induction hypothesis, $((t; \vec{m}; \bar{B} \parallel Q') : \diamond)$. First normalizing the derivation of this judgment and then applying Lemma 17 results in $((t; \vec{m}; \bar{B} \parallel Q) : \diamond)$.

Proof case, (Redn New): In this case, $P = (\mathsf{new}(k{:}U); P' \mid R)$, $k \notin fv(\vec{n}, R)$, $\vec{m} = (\vec{n}, k)$, $\bar{B} = \bar{A}$ and $Q = P' \mid R$. Let $E' = (E, k{:}U, \mathsf{fresh}(k))$ and $\bar{C}' = (\bar{C}, k{:}U, \mathsf{fresh}(k))$. Because $\mathcal{D}'$ is subnormal, it ends in (Par) preceded by (New). Therefore $(\bar{C}' \vdash P')$, U is generative and $(\bar{C} \vdash R)$. Then $(\bar{C}' \vdash P' \mid R = Q)$, by weakening and (Par). On the other hand, $(E' \vdash^{+} \bar{C}')$, by weakening, (Id) and (And).

Proof case, (Redn IO): In this case, $P = (\mathsf{out}\, N\, M \mid \mathsf{inp}\, N\, (x{:}U); P' \mid R)$, $\vec{m} = \vec{n}$, $\bar{B} = \bar{A}$ and $Q = P'\{x{\leftarrow}M\} \mid R$. Because $\mathcal{D}'$ is subnormal, it ends in (Par) preceded by (Out) and (In). Therefore $(\bar{C} \vdash M{:}\mathsf{Un})$, $(\bar{C}, x{:}\mathsf{Un} \vdash P')$ and $(\bar{C} \vdash R)$. Let $\bar{C}' = (\bar{C}, M{:}\mathsf{Un})$. From $(\bar{C}, x{:}\mathsf{Un} \vdash P')$ we obtain $(\bar{C}' \vdash P'\{x{\leftarrow}M\})$, by substitutivity. From $(\bar{C}' \vdash P'\{x{\leftarrow}M\})$ and $(\bar{C} \vdash R)$, we obtain $(\bar{C}' \vdash P'\{x{\leftarrow}M\} \mid R = Q)$, by weakening and (Par). On the other hand, cutting $(E \vdash^+ \bar{C})$ with $(\bar{C} \vdash M{:}\mathsf{Un})$ results in $(E \vdash^+ \bar{C}')$.

Proof case, (Redn Decrypt): In this case, $P = (\mathsf{decrypt}\ \{M\}_K\ \mathsf{is}\ \{x{:}T\}_K; P' \mid R)$, $\vec{m} = \vec{n}$, $B = \bar{A}$ and $Q = P'\{x{\leftarrow}M\} \mid R$. Because $\mathcal{D}'$ is subnormal, it ends in (Par) preceded by either (Decrypt) or (Decrypt Un). By inverting (Par), we obtain $(\bar{C} \vdash R)$. *Subcase (Decrypt):* In this subcase, we get $(\bar{C} \vdash \{M\}_K{:}\mathsf{Un}, K{:}\tau\text{-}\mathsf{Key}(\vec{N}'))$ and $(\bar{C}, x{:}\tau\text{-}\mathsf{Auth}(K, \vec{N}') \vdash P')$. From $(\bar{C} \vdash \{M\}_K{:}\mathsf{Un}, K{:}\tau\text{-}\mathsf{Key}(\vec{N}'))$ we get $(\bar{C} \vdash M{:}\tau\text{-}\mathsf{Auth}(K, \vec{N}'))$, by Lemma 15(a). From this point on, we proceed as in proof case (Redn IO). *Subcase (Decrypt Un):* In case, we have $(\bar{C} \vdash \{M\}_K{:}\mathsf{Un}, K{:}\mathsf{Un})$ and $(\bar{C}, x{:}\mathsf{Un} \vdash P')$. From $(\bar{C} \vdash \{M\}_K{:}\mathsf{Un}, K{:}\mathsf{Un})$ it follows that $(\bar{C} \vdash M{:}\mathsf{Un})$, by Lemma 15(b). From this point on, we proceed as in proof case (Redn IO).

Proof case, (Redn Untag): Similar to proof case (Redn Decrypt), using Lemma 15(c) and (d).

Proof cases, (Redn Split) and (Redn Match): Similar to proof case (Redn Decrypt), using Lemma 15(e) and (f).

Proof case, (Redn Clock): In this case, $P = (\mathsf{clock}(x{:}T); P' \mid R)$, $\vec{m} = \vec{n}$, $\bar{B} = \bar{A}$ and $Q = P'\{x{\leftarrow}t\} \mid R$. Because $\mathcal{D}'$ is subnormal, it ends in (Par) preceded by (Clock). Therefore $(\bar{C}, x{:}\mathsf{Un}, \mathsf{now}(x) \vdash P')$ and $(\bar{C} \vdash R)$. Let $\bar{C}' = (\bar{C}, t{:}\mathsf{Un}, \mathsf{now}(t))$. From $(\bar{C}, x{:}\mathsf{Un}, \mathsf{now}(x) \vdash P')$ we obtain $(\bar{C}' \vdash P'\{x{\leftarrow}t\})$, by substitutivity. From $(\bar{C}' \vdash P'\{x{\leftarrow}t\})$ and $(\bar{C} \vdash R)$, we obtain $(\bar{C}' \vdash P'\{x{\leftarrow}t\} \mid R = Q)$, by weakening and (Par). On the other hand, we obtain $(E \vdash^+ \bar{C}')$ from $(E \vdash^+ \bar{C})$, by (Time), (Id) and (And).

Proof case, (Redn Begin): We have $P = (\mathsf{begin!}(M); P' \mid R)$, $\vec{m} = \vec{n}$, $\bar{B} = (\bar{A}, \mathsf{end}(M))$ and $Q = P' \mid R$. Because $\mathcal{D}'$ is subnormal, it ends in (Par) preceded by (Begin). Therefore $(\bar{C}, \mathsf{end}(M) \vdash P')$ and $(\bar{C} \vdash R)$. Let $\bar{C}' = (\bar{C}, \mathsf{end}(M))$ and $E' = (E, \mathsf{end}(M))$. By weakening and (Par), we obtain $(\bar{C}' \vdash P' \mid R = Q)$. From $(E \vdash^+ \bar{C})$, we obtain $(E' \vdash^+ \bar{C}')$, by weakening, (Id) and (And). □

Let $(E \vdash' P)$ iff E is basic and there exists $\bar{A}$ such that $(E \vdash^+ \bar{A})$ and $(\bar{A} \vdash P)$ has a subnormal derivation. Let $P \xrightarrow{\sigma} Q$ iff $(t; \vec{n}; \bar{A} \mid\mid P) \xrightarrow{\sigma} (t+1; \vec{n}; \emptyset \mid\mid Q)$ for some $t, \vec{n}, \bar{A}$.

Lemma 19 (Tick Preserves Typing). *If $(E \vdash' P)$ and $P \xrightarrow{\sigma} Q$, then $(step^+(E) \vdash' Q)$.*

Proof By induction on $(P \xrightarrow{\sigma} Q)$'s derivation. Suppose $(E \vdash' P)$ and $P \xrightarrow{\sigma} Q$. By definition of $\vdash'$, E is a basic environment and there exist $\bar{A}, \mathcal{D}$ such that $(E \vdash^+ \bar{A})$ and $\mathcal{D} \rhd (\bar{A} \vdash P)$ is subnormal. From $(E \vdash^+ \bar{A})$ we obtain $(step^+(E) \vdash^+ step(\bar{A}))$, by step invariance.

Proof case, (Tick Par): In this case, $P = P_1 \mid P_2$, $(P_1 \xrightarrow{\sigma} Q_1)$, $(P_2 \xrightarrow{\sigma} Q_2)$ and $Q = Q_1 \mid Q_2$. Because $\mathcal{D}$ is subnormal, its last rule is (Par) and we obtain that $(E \vdash' P_1)$ and $(E \vdash' P_2)$. Then, by induction hypothesis, $(step^+(E) \vdash' Q_1)$ and $(step^+(E) \vdash' Q_2)$. By definition of $\vdash'$, this means that $(step^+(E) \vdash^+ \bar{B}_1)$, $(\bar{B}_1 \vdash Q_1)$, $(step^+(E) \vdash^+ \bar{B}_2)$ and $(\bar{B}_2 \vdash Q_2)$ for some $\bar{B}_1, \bar{B}_2$. Then $(step^+(E) \vdash^+ \bar{B}_1, \bar{B}_2)$, by (And). By Lemma 4, $(\bar{B}_1, \bar{B}_2 \vdash \diamond)$. Therefore, we can use weakening and (Par) to obtain $(\bar{B}_1, \bar{B}_2 \vdash Q)$. Then $(step^+(E) \vdash' Q)$, by definition of $\vdash'$.

Proof case, (Tick Crack): In this case, $P = (\mathsf{st\text{-}secret}(K) \mid \mathsf{crack}\ \{M\}_K\ \mathsf{is}\ \{x{:}T\}_{y{:}U}; P')$ and $Q = P'\{x,y{\leftarrow}M,K\}$. Because $\mathcal{D}$ is subnormal, it ends in (Par) preceded by (Crack). Inverting these rules gives $(\overline{A} \vdash \mathsf{st\text{-}secret}(K), \{M\}_K{:}\mathsf{Un})$ and $(\mathit{step}(\overline{A}), x{:}\mathsf{Un}, y{:}\mathsf{Un} \vdash P')$. Let $\overline{A'} = (\mathit{step}(\overline{A}), M{:}\mathsf{Un}, K{:}\mathsf{Un})$. Applying substitutivity to $(\mathit{step}(\overline{A}), x{:}\mathsf{Un}, y{:}\mathsf{Un} \vdash P')$ results in $(\overline{A'} \vdash P'\{x,y{\leftarrow}M,K\} = Q)$. Cutting $(E \vdash^+ \overline{A})$ with $(\overline{A} \vdash \mathsf{st\text{-}secret}(K), \{M\}_K{:}\mathsf{Un})$ results in $(E \vdash^+ \mathsf{st\text{-}secret}(K), \{M\}_K{:}\mathsf{Un})$. Then $(\mathit{step}^+(E) \vdash^+ \mathsf{public}(K), \{M\}_K{:}\mathsf{Un})$, by step invariance. Because $\mathit{step}^+(E)$ is basic, $(\mathit{step}^+(E) \vdash^+ \mathsf{public}(K))$'s derivation cannot end in (Id), thus ends in (Public), thus $(\mathit{step}^+(E) \vdash^+ K{:}\mathsf{Un})$. From $(\mathit{step}^+(E) \vdash^+ K{:}\mathsf{Un}, \{M\}_K{:}\mathsf{Un})$ we obtain $(\mathit{step}^+(E) \vdash^+ M{:}\mathsf{Un})$, by Lemma 15(b). Then $(\mathit{step}^+(E) \vdash^+ \mathit{step}(\overline{A}), M{:}\mathsf{Un}, K{:}\mathsf{Un} = \overline{A'})$, by (And). From $(\mathit{step}^+(E) \vdash^+ \overline{A'})$ and $(\overline{A'} \vdash Q)$ it follows that $(\mathit{step}^+(E) \vdash' Q)$, by definition of $\vdash'$.

Proof case, (Tick Remain), $P = !P'$: In this case $P = Q = !P'$. Because $\mathcal{D}$ is subnormal, it can only end in (Repl). Therefore $(\mathit{step}(\overline{A}) \vdash P')$. Because the *step*-function is idempotent, it is also the case that $(\mathit{step}(\mathit{step}(\overline{A})) \vdash P')$. Therefore, $(\mathit{step}(\overline{A}) \vdash Q)$, by (Repl). From $(\mathit{step}^+(E) \vdash^+ \mathit{step}(\overline{A}))$ and $(\mathit{step}(A) \vdash Q)$ we obtain $(\mathit{step}^+(E) \vdash' Q)$, by definition of $\vdash'$.

Proof case, (Tick Remain), $P = (\mathsf{out}\ N\ M)$: In this case $P = Q = (\mathsf{out}\ N\ M)$. Because $\mathcal{D}$ is subnormal, it can only end in (Out). Therefore $(\overline{A} \vdash N{:}\mathsf{Un}, M{:}\mathsf{Un})$. Cutting $(E \vdash^+ \overline{A})$ with $(\overline{A} \vdash N{:}\mathsf{Un}, M{:}\mathsf{Un})$ results in $(E \vdash^+ N{:}\mathsf{Un}, M{:}\mathsf{Un})$. Then $(\mathit{step}^+(E) \vdash^+ N{:}\mathsf{Un}, M{:}\mathsf{Un})$, by step invariance. On the other hand, $(N{:}\mathsf{Un}, M{:}\mathsf{Un} \vdash \mathsf{out}\ N\ M = Q)$. From $(\mathit{step}^+(E) \vdash^+ N{:}\mathsf{Un}, M{:}\mathsf{Un})$ and $(N{:}\mathsf{Un}, M{:}\mathsf{Un} \vdash Q)$ we get $(\mathit{step}^+(E) \vdash' Q)$, by definition of $\vdash'$.

Proof case, (Tick Remain), $P = \mathsf{public}(M)$ or $P = \mathsf{lt\text{-}secret}(M)$: In this case $P = Q = B$ for some B such that $\mathit{step}(B) = B$. Cutting $(E \vdash^+ \overline{A})$ with $(\overline{A} \vdash B)$ results in $(E \vdash^+ B)$. By step invariance, we get $(\mathit{step}^+(E) \vdash^+ B)$. On the other hand $(B \vdash B = Q)$. From $(\mathit{step}^+(E) \vdash^+ B)$ and $(B \vdash Q)$ we get $(\mathit{step}^+(E) \vdash' Q)$, by definition of $\vdash'$.

Proof case, (Tick Expire): In this case, $Q = \mathbf{0}$. By (And), $(\mathit{step}^+(E) \vdash^+)$. By (Zero), $(\vdash Q)$. Then $(\mathit{step}^+(E) \vdash' Q)$, by definition of $\vdash'$. □

Theorem 1 (Type Preservation). *If* $((t;\vec{n};\overline{A} \mid\mid P) : \diamond)$ *and* $(t;\vec{n};\overline{A} \mid\mid P) \Rightarrow (s;\vec{m};\overline{B} \mid\mid Q)$, *then* $((s;\vec{m};\overline{B} \mid\mid Q) : \diamond)$.

Proof By induction on the length of $\Rightarrow$. Suppose $\mathcal{D} \triangleright ((t;\vec{n};\overline{A} \mid\mid P))$ is normal and $(t;\vec{n};\overline{A} \mid\mid P) \Rightarrow (s;\vec{m};\overline{B} \mid\mid Q)$.

Proof case, $(t;\vec{n};\overline{A} \mid\mid P) \rightarrow (t;\vec{k};\overline{C} \mid\mid R) \Rightarrow (s;\vec{m};\overline{B} \mid\mid Q)$: By Lemma 18, $(t;\vec{k};\overline{C} \mid\mid R) : \diamond$. Then $(s;\vec{m};\overline{B} \mid\mid Q) : \diamond$, by induction hypothesis.

Proof case, $(t;\vec{n};\overline{A} \mid\mid P) \xrightarrow{\sigma} (t+1;\vec{n};\emptyset \mid\mid R) \Rightarrow (s;\vec{m};\overline{B} \mid\mid Q)$: Because $\mathcal{D}$ is normal, there exists an environment $E = (\vec{n}{:}\vec{T}, \mathsf{end}\{\vec{M}\}, \mathsf{fresh}\{\vec{N}\}, \mathsf{now}(t))$ such that $(E \vdash' P)$. Then $(\mathit{step}^+(E) \vdash' R)$, by Lemma 19. By definition of the step^+-function, $\mathit{step}^+(E) = (\vec{n}{:}\mathit{step}(\vec{T}), \mathsf{now}(t+1))$. From $\mathit{step}^+(E) = (\vec{n}{:}\mathit{step}(\vec{T}), \mathsf{now}(t+1))$ and $(\mathit{step}^+(E) \vdash' R)$ it follows that $((t+1;\vec{n};\emptyset \mid\mid R) : \diamond)$, by (Good State). Then, by induction hypothesis, $(s;\vec{m};\overline{B} \mid\mid Q) : \diamond$. □

Lemma 20 (Message Typability). *If* $\mathit{fv}(M) = \vec{n}$, *then* $(\vec{n}{:}\mathsf{Un} \vdash M{:}\mathsf{Un})$.

Proof By induction on the structure of M, using (Tag Un), (Encrypt Un), (Sub Pair Un) and (Sub Ok Un). □

Lemma 21 (Opponent Typability). *If O is an opponent process and $fv(O) = \vec{n}$, then $(\vec{n}{:}\mathsf{Un} \vdash O)$.*

Proof By induction on the structure of P, using Lemma 20 and rules (Untag Un), (Decrypt Un), (Split Un) and (Match Un). □

Lemma 22 (Safety). *If $\vec{n}$ are distinct, $\vec{T}$ are generative and $(\vec{n}{:}\vec{T} \vdash P)$, then P is safe.*

Proof Suppose $\vec{n}$ are distinct, $\vec{T}$ are generative and $(\vec{n}{:}\vec{T} \vdash P)$. By Lemma 3, $fv(P) \subseteq \vec{n}$. We may assume that new-bound names in P are distinct from $\vec{n}$.

For secrecy, suppose towards a contradiction that $(t; fv(P); \emptyset \mid\mid P) \Rightarrow (s; \vec{m}; \overline{A} \mid\mid R)$ where $R = (\mathsf{public}(N) \mid \tau\text{-}\mathsf{secret}(M) \mid \mathsf{out}\ N\ M \mid Q)$. Because $fv(P) \subseteq \vec{n}$, it is then also the case that $(t; \vec{n}; \emptyset \mid\mid P) \Rightarrow (s; \vec{m}; A \mid\mid R)$. By (Good State), $(t; \vec{n}; \emptyset \mid\mid P) : \diamond$. By type preservation, $(s; \vec{m}; A \mid\mid R) : \diamond$. By normalization, $\mathcal{D} \triangleright ((s; \vec{m}; A \mid\mid R) : \diamond)$ for some normal $\mathcal{D}$. By inverting $\mathcal{D}$'s last rules and cut, we obtain a basic environment E such that $(E \vdash^+ \tau\text{-}\mathsf{secret}(M))$ and $(E \vdash^+ M{:}\mathsf{Un})$. Because E is basic, the last rule of $(E \vdash \tau\text{-}\mathsf{secret}(M))$'s derivation can only be (Secret), thus $(E \vdash^+ M : \tau\text{-}\mathsf{Secret})$. Then, by (And), $(E \vdash^+ M : \tau\text{-}\mathsf{Secret}, M : \mathsf{Un})$, in contradiction to Lemma 14.

For authenticity, suppose $(t; fv(P); \emptyset \mid\mid P) \Rightarrow (s; \vec{m}; \overline{A} \mid\mid R)$ where $R = (\mathsf{end}(M) \mid Q)$. Because $fv(P) \subseteq \vec{n}$, it is then also the case that $(t; \vec{n}; \emptyset \mid\mid P) \Rightarrow (s; \vec{m}; \overline{A} \mid\mid R)$. By (Good State), $(t; \vec{n}; \emptyset \mid\mid P) : \diamond$. By type preservation, $(s; \vec{m}; \overline{A} \mid\mid R) : \diamond$. By normalization, $\mathcal{D} \triangleright ((s; \vec{m}; \overline{A} \mid\mid R) : \diamond)$ for some normal $\mathcal{D}$. By inverting $\mathcal{D}$'s last rules and cut, we obtain an environment $E = (\vec{m}{:}\vec{T}, \mathsf{end}\{\vec{L}\}, \mathsf{fresh}\{\vec{N}\})$ such that $\overline{A} = \mathsf{end}\{\vec{L}\}$ and $(E \vdash^+ \mathsf{end}(M))$. The only possible reason for $(E \vdash^+ \mathsf{end}(M))$ is (Id). Therefore $\mathsf{end}(M) \in E$. But this is only possible if $\mathsf{end}(M) \in \mathsf{end}\{\vec{L}\} = \overline{A}$. □

Theorem 2 (Robust Safety). *If $\vec{n}$ are distinct, $(\vec{n}{:}\vec{T} \vdash \mathsf{public}(\vec{n}))$ and $(\vec{n}{:}\vec{T} \vdash P)$, then P is robustly safe.*

Proof Suppose $\vec{n}$ are distinct, $(\vec{n}{:}\vec{T} \vdash \mathsf{public}(\vec{n}))$ and $(\vec{n}{:}\vec{T} \vdash P)$. From $(\vec{n}{:}\vec{T} \vdash \mathsf{public}(\vec{n}))$ it follows that $(\vec{n}{:}\vec{T} \vdash \vec{n}{:}\mathsf{Un})$. Then $\vec{T} \leq \mathsf{Un}$, by Lemma 12. Let O be an opponent process with $fv(O) \subseteq (\vec{n}, \vec{m})$. By opponent typability and weakening, $(\vec{n}{:}\mathsf{Un}, \vec{m}{:}\mathsf{Un} \vdash O)$. By Lemma 6, $(\vec{n}{:}\vec{T}, \vec{m}{:}\mathsf{Un} \vdash O)$. By weakening, $(\vec{n}{:}\vec{T}, \vec{m}{:}\mathsf{Un} \vdash P)$. Then $(\vec{n}{:}\vec{T}, \vec{m}{:}\mathsf{Un} \vdash P \mid O)$, by (Par). Moreover, $\vec{T}$ are generative, because subtypes of Un are. Then $P \mid Q$ is safe, by Lemma 22. □