Authenticity by Tagging and Typing [*]

Michele Bugliesi Riccardo Focardi Matteo Maffei
Università Ca'Foscari di Venezia
Dipartimento di Informatica
Via Torino 155, 30172 Mestre (VE), Italy
{bugliesi,focardi,maffei}@dsi.unive.it

ABSTRACT

We propose a type and effect system for *authentication* protocols built upon a tagging scheme that formalizes the intended semantics of ciphertexts. The main result is that the validation of each component in isolation is provably sound and *fully compositional*: if all the protocol participants are independently validated, then the protocol as a whole guarantees authentication in the presence of Dolev-Yao intruders. The highly compositional nature of the analysis makes it suitable for multi-protocol systems, where different protocols might be executed concurrently.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Program Analysis*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security, Verification

Keywords

Static Analysis, Authentication, Process Calculi

1. INTRODUCTION

Authentication protocols are security protocols whose purpose is to enable two entities to achieve mutual (and reliable!) agreement on some piece of information, typically the identity of the other party, its presence, the origin of a message, its intended destination. Achieving the intended agreement guarantees is subtle because they typically are the result of the encryption/decryption of (sequences of) messages composed of different parts, with each part providing a "piece" of the authentication guarantee. To illustrate, consider the following example:

$$\begin{array}{lcll} B & \rightarrow & A: & n \\ A & \rightarrow & B: & \{B,m,n\}_{\mathsf{Sign_A}} \end{array}$$

Bob, the initiator, sends to Alice a random challenge (i.e., a nonce) n; Alice, the responder, signs the challenge together with a message m and Bob's identifier B; the protocol completes with the equality check on the nonce performed by B after decrypting the message with A's public key. The aim of the protocol is to guarantee to B that *m is authentic from A*, i.e., that m has been freshly sent by A to B. Each message component of the encrypted packet $\{B,m,n\}_{\mathsf{Sign_A}}$ has a specific purpose in the protocol: the nonce n guarantees that the packet is fresh (it is not a reply of an old protocol session since n is used only once); the identifier B specifies the intended receiver of m; the signature guarantees that A is the sender. Collectively, the exchange of the three components provides the following *agreement* [18, 23] property: (i) the two participants agree on each other's identity, namely, B is guaranteed that its responder is A, as it appears to be, and dually, A knows that the initiator is indeed B; (ii) the two parties agree on the origin and destination of the message m, i.e. that it was originated by A and intended for B.

In [10] we have proposed a static method for the analysis of authentication properties of the kind outlined above. Here, we extend our results in several ways.

- We make our methodology applicable to a substantially wider class of protocols, based on both symmetric and asymmetric cryptography, and on diverse challenge-response mechanisms.
- We provide authentication guarantees in the presence of a wider class of attacks, including attacks based on the opponent acting as an honest principal. Thus our analysis encompasses the case of opponents provided with their own private, public and long-term keys.
- We extend our analysis to the verification of *message authentication* properties in addition to entity authentication.
- Finally, we formulate our approach in terms of a type-and-effect system; this provides our system with a more solid meta-theory, based on consolidated type-theoretic foundations, and enables us to draw more precise comparisons with related work on the subject, specifically with the work by Gordon and Jeffrey [11, 12].

Our present approach is based on the same methodology as the one proposed by [11, 12], namely: (1) specify properties by annotating an executable specification of the protocol with correspondence assertions; (2) annotate the protocol with suitable types (and tags);

[*]Work partially supported by EU Contract IST-2001-32617 "Models and Types for Security in Mobile Distributed Systems" (MyThS).

(3) verify the assertions by running a type checker. Technically, however, the technique we employ is different, as we rely on tags to annotate the ciphertexts exchanged in the protocol to assist the typed analysis of the protocol. This is best illustrated with an example.

Consider again the simple authentication protocol we illustrated earlier. As we argued, the authenticity properties of the protocol are the result of the combined effect of specific guarantees conveyed by each of the components of the message $\{B,m,n\}_{\mathsf{Sign_A}}$. The system of [12] captures these guarantees by assigning the following type to the private key that signs the message:

$$\begin{array}{ll} Key(& B : \mathsf{Principal}, \\ & m : \mathsf{Payload}, \\ & n : \mathsf{Public\ Response}[\mathsf{end}\ A\ \mathit{sending}\ m\ \mathit{to}\ B] \quad) \end{array}$$

The structure of this dependent type renders the intended dependencies among the message components: in particular, the nonce type provides a complete specification of the role of the nonce in the protocol. The safety proof for the protocol is then, essentially, a consequence of the typing rules guaranteeing that the protocol participants manipulate the nonce according to this intended usage. The fairly rich structure of the key and nonce types makes the approach very flexible, and expressive. On the other hand, there appear to be trade-offs. First, step (2) of the method (see above) may become rather complex, for choosing the correct types of nonces and keys may turn out to be non-trivial. Secondly, the degree of compositionality of the analysis seems to be slightly undermined by the very structure of the types, which essentially encode much of the structure of the protocol itself. As a consequence, it seems hardly possible to factor the authenticity properties proved for the protocol into corresponding properties established locally for each of the principals, independently of the context.

Our approach is different. We render the inter-dependencies among message components directly on terms (rather than on their types), by imposing a richer, *tagged*, structure to our terms. We identify "minimal" set of authentication *patterns* (based on corresponding forms of nonce handshakes), and make explicit in the encrypted messages the pattern they correspond to. For example, message $\{B,m,n\}_{\mathsf{Sign_A}}$ is tagged as $\{\mathsf{Id}(B),\mathsf{Auth}(m),\mathsf{Verif}(n)\}_{\mathsf{Sign_A}}$ to signal that the nonce n is authenticating m to the verifier B. The tags are dynamically identified by the recipient of a message, upon decryption, and employed to achieve/provide authentication guarantees. Intuitively, the information associated through dependent types in the Gordon-Jeffrey type system, is conveyed here in the more complex structure of our (tagged) terms. This allows us to use simple types to just enforce the secrecy of keys and secret challenges, and simple effects to reason about authentication.

The advantages of our approach may be summarized as follows.

It is fully compositional: since the uniform use of tags is imposed by the typing rules, each party may be checked in isolation, i.e., using an independent typing environment. Indeed, all parties do share the same typing assumptions for keys, but such types only convey information on the principals holding/sharing the keys and, more importantly, this information is independent of the specific protocol that the entities will be running (hence, we do not need to include in the type assumptions for keys any information about the structure of the messages that will be encrypted with such keys). This is different from the Gordon-Jeffrey approach: as we noted above, in [12] the types of the shared keys encode information on the protocol steps, so that different protocols between the same parties require different typings for the same keys (or else, different keys).

This strong form of compositionality allows us to safely mix different protocols once their sequential components are type-checked; thus, our tagging discipline naturally scales to multi-protocol settings. Furthermore, the fact that tags correspond to a small set of a-priori selected patterns, makes the type system quite simple and easy to use; the human effort required is very small. Finally, even though the set of authentication patterns we have selected is small, it seems to be expressive enough to capture many of the protocols in literature; this gives also new insights on which are the basic mechanisms for guaranteeing authentication. We remark that our safety results rely critically on the assumption that the messages exchanged in the protocol are tagged: hence our tags play a static as well a dynamic role, and the safety theorem assumes that the semantics of protocols is itself tagged. While this may be seen as a limitation, our tagging mechanism turns out to be less demanding than those employed to resolve message ambiguities in many existing protocol implementations and protocol analysis techniques (cf. Section 5).

Plan of the paper The rest of the paper is organized as follows: Section 2 gives a brief outline of the ρ-spi calculus and its operational semantics. Section 3 summarizes our type and effect system and its main properties. In Section 4 we analyze the Splice/AS Protocol and in Section 5 we conclude with final remarks and a discussion of other related work.

2. ρ-spi calculus

The ρ-spi calculus , originally proposed in [10], derives from the spi calculus [2], and inherits many of the features of *Lysa* [5], a version of the spi calculus proposed for the analysis of authentication protocols. ρ-spi differs from both calculi in several respects: it incorporates the notion of tagged message exchange from [9], it provides new authentication-specific constructs, and offers primitives for declaring process identities and keys. In this paper, we extend the calculus with primitives for asymmetric cryptography (key declaration, encryption and decryption).

The syntax is reported in Table 1 and described below. We presuppose two countable sets: $\mathcal{N}$ of names and $\mathcal{V}$ of variables. We reserve k,m,n for names and x,y,z for variables, with a ranging over both names and variables. Identities $\mathcal{ID}$ are a subset of names and are ranged over by I and J. Identities are further partitioned into *trusted principals* $I_{\mathcal{P}}$, ranged over A and B, and *enemies* $I_{\mathcal{E}}$, ranged over by E. Both names and variables can be tagged, noted $C(a)$. Tags, denoted by C, are a special category of names. The pair composed by a public key and the corresponding private one is noted by $\mathsf{Pub}(m)$, $\mathsf{Priv}(m)$, similarly to [2][1].

Processes (or *protocols*), ranged over by P,Q are the parallel composition of principals. Each principal is a sequential process associated with an identity I, noted $I \triangleright S$. The replicated form $I \triangleright !S$ indicates an arbitrary number of copies of $I \triangleright S$. In order to allow the sharing of keys among principals, we provide ρ-spi with let-bindings: $\mathsf{let}\ k = \ \mathsf{sym\text{-}key}(I_1,I_2).P$ declares (and binds) the long-term key k shared between I_1 and I_2 in the scope P. Similarly, $\mathsf{let}\ k = \ \mathsf{asym\text{-}key}(I).P$ declares, and binds in the scope P, the key pair $\mathsf{Pub}(k)$, $\mathsf{Priv}(k)$ associated to I.

Sequential processes may never fork into parallel components: this assumption helps assign unique identities to (sequential) processes, and involves no significant loss of expressive power as protocol principals are typically specified as sequential processes, possibly sharing some long-term keys. The sequential process $\mathbf{0}$ is the null process that does nothing, as usual. Process $\mathsf{new}(n).S$ generates

[1]In the rest of the paper, we will use the following notation convention: $\overline{\mathsf{Pub}} = \mathsf{Priv}$ and vice-versa.

Table 1 The syntax of ρ-spi calculus.

Notation: C over tags: $\{\mathsf{Claim}, \mathsf{Claim?}, \mathsf{Verif}, \mathsf{Verif?}, \mathsf{Id}, \mathsf{Auth}\}$, a over names and variables.

$\mathcal{M}$ ::= *Patterns*	
m, n, k	names
x, y, z	variables
$C(a)$	tagged data
$\mathsf{key}(a)$	key($key \in \{\mathsf{Pub}/\mathsf{Priv}\}$)

P, Q ::= *Processes*	
$I \triangleright S$	(principal)
$I \triangleright !S$	(replication)
$P \mid Q$	(composition)
$\mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1, I_2).P$	(symmetric-key assignment)
$\mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P$	(asymmetric-key assignment)

S ::= *Sequential processes*	
$\mathbf{0}$	(nil)
$\mathsf{new}(n).S$	(restriction)
$\mathsf{in}(\mathcal{M}_1, \ldots, \mathcal{M}_n).S$	(input)
$\mathsf{out}(\mathcal{M}_1, \ldots, \mathcal{M}_n).S$	(output)
$\mathsf{encrypt}\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}_{\mathcal{M}_0}\ \mathsf{as}\ x.S$	(symmetric encryption)
$\mathsf{encrypt}\{\|\mathcal{M}_1, \ldots, \mathcal{M}_n\|\}_{\mathcal{M}_0}\ \mathsf{as}\ x.S$	(asymmetric encryption)
$\mathsf{decrypt}\ x\ \mathsf{as}\ \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}_{\mathcal{M}_0}.S$	(symmetric decryption)
$\mathsf{decrypt}\ x\ \mathsf{as}\ \{\|\mathcal{M}_1, \ldots, \mathcal{M}_n\|\}_{\mathcal{M}_0}.S$	(asymmetric decryption)
$\mathsf{run}(I_1, I_2, \mathcal{M}).S$	(run)
$\mathsf{commit}(I_1, I_2, \mathcal{M}).S$	(commit)

a fresh name n local to S. We presuppose a unique (anonymous) public channel, the network, from/to which all principals, including intruders, read/send messages. Similarly to *Lysa*, our input primitive may (atomically) test part of the message read, by pattern-matching. If the input message matches the pattern, then the variables occurring in the pattern are bound to the remaining subpart of the message; otherwise the message is not read at all. For example, process '$\mathsf{in}(\mathsf{Claim}(x)).P$' may only read messages of the form $\mathsf{Claim}(M)$, binding x to M in P. Encryption just binds x to the encrypted message, while decryption checks if the message contained in x matches the form $\{M_1, \ldots, M_n\}_{M_0}$ (or $\{|M_1, \ldots, M_n|\}_{M_0}$), i.e., is a tuple of n messages encrypted with the appropriate key. Only in this case x is decrypted and the variables in the patterns $\mathcal{M}_1, \ldots, \mathcal{M}_n$ get bound to the decrypted messages. Similarly to the input primitive, decryption also may test part of the decrypted messages by pattern-matching mechanism. Finally, the primitives $\mathsf{run}(I_1, I_2, \mathcal{M}).S$ and $\mathsf{commit}(I_1, I_2, \mathcal{M}).S$ declare that the sequential process I_1 is starting and committing respectively, a protocol session with I_2 for authenticating message $\mathcal{M}$. These constructs are used to check the *correspondence assertions* [24].

It is important to note that we make a distinction between *static* terms, or *patterns*, and dynamic terms, or *messages*. The former, noted $\mathcal{M}$, define the set of *syntactically legal* terms; the latter, noted M, define the set of terms that may arise at run time. The difference is that messages may have (nested) encryptions, while patterns may not: notice, to this regard, that in the syntax of processes, encryptions may only be formed by means of the $\mathsf{encrypt}$ prefix.

Example 1. Let us consider a (flawed) simplification of the protocol presented in the Introduction, obtained by eliminating the nonce. (we implement signature by encrypting with the private key.)

$$A \quad \rightarrow \quad B: \qquad \{B, m\}_{\mathsf{Priv}(k_A)}$$

The narration of the protocol in ρ-spi is reported in Table 2; after declaring the key pair for A, an unbounded number of instances of A as initiator and an unbounded number of instances of B as responder are run in parallel. The Initiator generates a new message m, declares the start of the session with the responder, signs m together with the responder identifier and sends the obtained ciphertext on the network. The Responder reads a message from the network and tries to decrypt it with the public key of the initiator and checks that its own identifier is the first component of the plaintext. If this is the case, it commits on the received message x.

Operational Semantics. We define the operational semantics of ρ-spi in terms of *traces*, after [7]. A trace is a possible sequence of *actions* performed by a process. Each process primitive has an associated action and we denote with *Act* the set of all possible actions. The dynamics of the calculus is is formalized by means of a transition relation between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in Act^*$ is a trace, P is a (closed) process. Each transition $\langle s, P\rangle \longrightarrow \langle s :: \alpha, P'\rangle$ simulates one computation step in P and records the corresponding action in the trace.

Principals do not directly synchronize with each other. Instead, they may receive from the unique channel an arbitrary message M known by the environment, which models the Dolev-Yao intruder: the environment knows all the identity labels, the messages sent on the network, the content of ciphertexts whose decryption key is known, ciphertexts created by its knowledge and all the keys declared as owned by E together with all the public keys. Finally, it may create fresh names not appearing in the trace. The transition relation is given in detail in Appendix A.

Definition 1 (Traces) *The set $T(P)$ of traces of process P is the set of all the traces generated by a finite sequence of transitions from the configuration $\langle \varepsilon, P\rangle$: $T(P) = \{s \mid \exists P'$ s.t. $\langle \varepsilon, P\rangle \longrightarrow^* \langle s, P'\rangle\}$*

The notion of safety (similar to the one in [11]) formalizes the *correspondence* property of [24], also known as *agreement* [18].

Definition 2 (Safety) *A trace s is safe if and only if whenever $s = s_1 :: commit(B, A, M) :: s_2$, then $s_1 = s_1' :: run(A, B, M) :: s_1''$, and $s_1' :: s_1'' :: s_2$ is safe. A process P is safe if, $\forall s \in T(P), s$ is safe.*

Table 2 Simple Protocol in ρ-spi calculus

$$\begin{array}{rcl} Protocol & \triangleq & \mathsf{let}\ k_A = \mathsf{asym\text{-}key}(A)\ .\ (A \triangleright !Initiator \mid B \triangleright !Responder) \\ Initiator & \triangleq & \mathsf{new}(m).\mathsf{run}(A,B,m).\mathsf{encrypt}\ \{|B,m|\}_{\mathsf{Priv}(k_A)}\ \mathsf{as}\ z.\mathsf{out}(z). \\ Responder & \triangleq & \mathsf{in}(z).\mathsf{decrypt}\ z\ \mathsf{as}\ \{|B,x|\}_{\mathsf{Pub}(k_A)}.\mathsf{commit}(B,A,x) \end{array}$$

Informally, a trace is *safe* if every $commit(B,A,M)$ is preceded by a distinct $run(A,B,M)$. This guarantees that whenever B is convinced of the identity of A sending M, then A has indeed started the protocol with B for authenticating M.

Example 2. To illustrate the semantics of the calculus and the notion of safety, let us consider again the protocol of Example 1. It suffers of the following standard replay attack, where E impersonates A by just replaying a previously intercepted message:

$$\begin{array}{lcll} A & \rightarrow & B: & \{B,m\}_{\mathsf{Priv}(k_A)} \\ E(A) & \rightarrow & B: & \{B,m\}_{\mathsf{Priv}(k_A)} \end{array}$$

Consider the ρ-spi calculus specification of Table 2. The attack mentioned above corresponds to the following execution trace of process $Protocol$:

$$\begin{array}{l} asym-key(k_A,A) :: new(m) :: \mathbf{run}(A,B,m) :: \\ encrypt\{|B,m|\}_{\mathsf{Priv}(k_A)} :: out(\{|B,m|\}_{\mathsf{Priv}(k_A)}) :: \\ in(\{|B,m|\}_{\mathsf{Priv}(k_A)}) :: decrypt\{|B,m|\}_{\mathsf{Priv}(k_A)} :: \mathbf{commit}(B,A,m) :: \\ in(\{|B,m|\}_{\mathsf{Priv}(k_A)}) :: decrypt\{|B,m|\}_{\mathsf{Priv}(k_A)} :: \mathbf{commit}(B,A,m) \end{array}$$

Notice that the same message $\{B,m\}_{\mathsf{Priv}(k_A)}$ is read twice by two different instances of the Responder. This causes two commits with just one run, thus making this trace unsafe. The presence of the nonce (as discussed in the Introduction) repairs the protocol avoiding this replay attack.

3. THE TYPE AND EFFECT SYSTEM

Authentication protocols are typically based on *challenge-response* schemes in which the knowledge of a secret key is proved by either encrypting or decrypting a fresh challenge using such a key. Here we consider authentication protocols that use *nonces* (number used only once) as fresh challenges. There are three possible kinds of nonce handshakes [12, 15]: in *Public-Out Secret-Home* (*POSH*), the nonce is sent as cleartext and received into a ciphertext, in *Secret-Out Public-Home* (*SOPH*) the opposite happens, while in *Secret-Out Secret-Home* (*SOSH*), the nonce is sent out and received back into (different) ciphertexts.

Our type and effect system is based on some basic notions and rules (Section 3.1), and generic rules for processes (Section 3.2). The core of the analysis collects rules for checking authentication based on all of the above described nonce handshakes (Section 3.3). All types, effects and typing rules involve judgments on messages, i.e. the dynamic terms arising at run time: this is required to prove Subject reduction for the type and effect system.

3.1 Basic Notions and Rules

The definitions of the types and effects as well as the rules for deriving basic judgments are in Table 3. The typing environment Γ is an ordered set of bindings between names/variables and types. Types regulate the use of terms in the authentication task. A long-term key shared between I and J has type $key_{sym}(I,J)$ and it can only be used by I and J. A name used for creating a key pair owned by I has type $key_{asym}(I)$; the private and public keys created by that name have type $key_{pub}(I)$ and $key_{priv}(I)$, respectively. While private keys can only be used by their owners, public keys are available to every principal. Every untagged term potentially known by the enemy has type Un. A nonce used by I and J in a $SOPH/SOSH$ for authenticating M has initially type $nonce(I,J,M)$. As explained in Section 3.3, when the nonce is sent back by I, the type is casted to Un, since the nonce may be sent as cleartext. Trusted principals can encrypt messages only subject to certain hypotheses: the type $enc(f)$ is assigned to ciphertexts which can be created under the hypothesis represented by the atomic effect f. This type is discussed in Section §3.2, where we also describe effects.

Well-Formedness conditions are mostly standard. The empty environment is well-formed (Env ∅). An environment Γ is well-formed (Env Name and Var) only if it does not contain multiple occurrences of the same name/variable a. Moreover, types may only depend on untrusted terms. Since identities have always type Un (cf. rule Identity), this condition just enforces that M has type Un in $nonce(I,J,M)$.

The typing rules for public and private keys are straightforward. The type and effect system is provided with two sub-typing rules, namely Enemy Knowledge, in Table 3 and Secret Nonce (discussed in section 3.4, Table 6). Enemy Knowledge characterizes the knowledge of the enemy: as mentioned in §2, the enemy may be provided with both symmetric and asymmetric long-term keys; furthermore, the enemy knows all the public keys and may learn the secret nonces intended to be shared with him. Hence, all the corresponding types are subtypes of Un.

3.2 Typing Processes

To ease the presentation of the process calculus, so far we have given an untyped version of the restriction. Indeed, the type of nonces needs to be explicitly indicated in the restriction primitive. We will write $\mathsf{new}(n : T)$, where T is either Un or $nonce(I,J,M)$.

Processes are typed according to the judgment $\Gamma \vdash P : e$. Intuitively, $\Gamma \vdash P : e$ means that the process P can be typed under the typing environment Γ and the hypotheses expressed by the effect e. Judgment $I;\Gamma \vdash S : e$, for the sequential process S executed by the entity I, has the same intuitive meaning.

Effects are multisets of the atomic effects reported in Table 3, and their intuitive meaning is explained below.

- The freshness of a public nonce n is expressed by $fresh(n)$ or $fresh(n,I,J,M)$, according to the type of the nonce: $fresh(n)$ is used in $POSH$ nonce handshakes, where n is generated as Un and sent in clear, while $fresh(n,I,J,M)$ is used for $SOPH/SOSH$, where n has type $nonce(I,J,M)$ and is sent encrypted. These atomic effects are added when the nonce is generated and are removed whenever a commit requires the freshness of n. The purpose is to guarantee that each nonce is checked at most once, i.e., there is at most one commit on each nonce.

- Sometimes an encryption represents the start of an authentication session (as in the simple protocol of Example 2). We require each of those encryptions to be preceded by a different run. The atomic effect $run(I,J,M)$ serves this purpose: it

Table 3 Types Definitions

a, b range over names and variables.

Types

T	$::=$	$key_{sym}(I,J)$	long-term key
		$key_{asym}(I)$	asymmetric component
		$key_{pub}(I)$	public key
		$key_{priv}(I)$	private key
		Un	untrusted
		$nonce(I,J,M)$	secret nonce
		$enc(f)$	ciphertext

Atomic Effects

f	$::=$	$\oslash$	empty effect
		$fresh(n)$	public nonce freshness
		$fresh(n,I,J,M)$	secret nonce freshness
		$run(I,J,M)$	run
		$in(M)$	input
		$dec\{M_1,\ldots,M_n\}_{M_0}$	decryption

Environment Well-Formedness

Env $\oslash$
$$\oslash \vdash \diamond$$

Env Name and Var
$$\frac{a \neq I \qquad a \notin dom(\Gamma) \qquad \Gamma \vdash \diamond \qquad T \text{ depends on } M \Rightarrow \Gamma \vdash M : Un}{\Gamma, a : T \vdash \diamond}$$

Projection
$$\frac{\Gamma, a : T, \Gamma' \vdash \diamond}{\Gamma, a : T, \Gamma' \vdash a : T}$$

Typing Rules for Messages

Identity
$$\Gamma \vdash I : Un$$

Public Key
$$\frac{\Gamma \vdash k : key_{asym}(I)}{\Gamma \vdash \mathsf{Pub}(k) : key_{pub}(I)}$$

Private Key
$$\frac{\Gamma \vdash k : key_{asym}(I)}{\Gamma \vdash \mathsf{Priv}(k) : key_{priv}(I)}$$

Subsumption
$$\frac{\Gamma \vdash N : T' \qquad \Gamma \vdash T' <: T}{\Gamma \vdash N : T}$$

Enemy Knoledge
$$\frac{T \in \{key_{sym}(E,I), key_{asym}(E), key_{priv}(E), key_{pub}(I), nonce(E,I,M)\}}{\Gamma \vdash T <: Un}$$

is added to the typing effects when a run primitive is typed, and it is removed when typing an encryption representing a new session between I and J exchanging M.

- Receiving M is recorded by the atomic effect $in(M)$ and decrypting the ciphertext $\{M_1,\ldots,M_n\}_{M_0}$ is recorded by the atomic effect $dec\{M_1,\ldots,M_n\}_{M_0}$. These effects are useful in the authentication rules (Table 5) to check that a message has been received/decrypted.

The Rules for judgments $\Gamma \vdash P : e$ and $I;\Gamma \vdash S : e$ are reported in Table 4 and described below. Key declarations are typed by Symmetric and Asymmetric Key: the key just created is inserted in Γ with the appropriate type. Replication typechecks the replication of sequential processes, while Par typechecks the parallel composition of two processes: the resulting effect is the the sum of the effects required for typing the components. Identity Assignment is straightforward: $\Gamma \vdash A \triangleright S : e$ only if $A;\Gamma \vdash S : e$, namely only if S can be executed by A under Γ and the hypotheses in e. Notice that sequential processes can be executed only by trusted principals: as discussed in §2, the enemy is implicitly modeled by the semantics. By Nil, the null process can be always typed. Run adds the atomic effect $run(A,I,M)$ to the typing effect of the continuation process. In Input, all the free variables in the input messages are inserted in Γ with type Un. The atomic effects $in(M_i)$ keep track of the received messages. By Output, a term can be sent on the network only if it is typable by Un.

The restriction may be typed by two different rules: New Name and New Secret Nonce. In the former, the nonce is intended to be sent as cleartext on the network, so n is typed with Un. Since the nonce is fresh, $fresh(n)$ is added to the typing effect of the following process. Similar reasoning applies also to the latter. The type $nonce(I,J,M)$ prevents the nonce to be sent as cleartext on the network: in fact, as mentioned above, only terms with type Un can be sent on the network.

Symmetric and Asymmetric Encrypt rely on the judgment $A;\Gamma \vdash \{M_1,\ldots,M_n\}_{M_0} : enc(f)$, which means that the ciphertext $\{M_1,\ldots,M_n\}_{M_0}$ can be encrypted by A under the typing environment Γ and the atomic effect f. Such an effect is either the empty effect or $run(A,I,M)$: as mentioned above, an encryption may represent the start of an authentication session, so the corresponding run must have been previously executed. Typing rules for $enc(f)$ are reported in Table 5 and discussed in Section 3.3.

As shown in Symmetric and Asymmetric Decrypt, when decrypting a ciphertext, the highest types are assigned to the free variables according to the encryption rule originating that ciphertext. The encryption rules are mutually exclusive and univocally determine these highest types. In practice, finding these types is trivial, since it amounts to giving type Un to all variables except the ones with tags $\mathsf{Claim}?, \mathsf{Verif}?$ which require type $nonce(I,J,M)$, as described in Section 3.3. $\Gamma_{\overline{A}}$ simulates the typing environment of the originator of the message, which possibly differs from Γ because of secret nonces casted to Un in *SOPH*/*SOSH* challenges (see Section 3.3 for more detail).

3.3 Typing Authentication

We continue our description of the typing rule by illustrating the typing of the nonce-handshakes. The authentication rules are in Table 5. In the following, $\widetilde{M}$ represents a sequence of terms; the order of the terms in a ciphertext is immaterial; finally, $\Gamma \vdash M_1,\ldots,M_n : T$ is a short for $\Gamma \vdash M_1 : T, \ldots, \Gamma \vdash M_n : T$.

POSH Nonce Hand-Shake

In a *POSH* nonce hand-shake, J creates a fresh nonce n with type Un (New Name) and sends it on the network. I receives it and creates a ciphertext by POSH Request for authenticating a mes-

Table 4 Typing Processes

In SYMMETRIC DECRYPT , $\Gamma \vdash K : key_{sym}(I,J) \Rightarrow A \in \{I,J\}$. In ASYMMETRIC DECRYPT , $\Gamma \vdash K : key_{priv}(I) \Rightarrow A = I$.
In SYMMETRIC and ASYMMETRIC DECRYPT, $\Gamma_{\overline{A}} = \Gamma[n : Un/n : nonce(I,A,M)]$, $\forall n,I,M$.

SYMMETRIC KEY
$$\frac{\Gamma, k : key_{sym}(I,J) \vdash P : e}{\Gamma \vdash \mathsf{let}\ k = \mathsf{sym\text{-}key}(I,J).P : e}$$

ASYMMETRIC-KEY
$$\frac{\Gamma, k : key_{asym}(I) \vdash P : e}{\Gamma \vdash \mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P : e}$$

REPLICATION
$$\frac{\Gamma \vdash A \triangleright S : e}{\Gamma \vdash A \triangleright !S : e}$$

PAR
$$\frac{\Gamma \vdash P : e_P \qquad \Gamma \vdash Q : e_Q}{\Gamma \vdash P|Q : e_P + e_Q}$$

IDENTITY ASSIGNMENT
$$\frac{A;\Gamma \vdash S : e}{\Gamma \vdash A \triangleright S : e}$$

NIL
$$\frac{\Gamma \vdash \diamond}{A;\Gamma \vdash \mathbf{0} : e}$$

RUN
$$\frac{A;\Gamma \vdash S : e + [run(A,I,M)]}{A;\Gamma \vdash \mathsf{run}(A,I,M).S : e}$$

INPUT
$$\frac{A;\Gamma, fv(M_1,\ldots,M_n) : Un \vdash S : e + [in(M_1) + \ldots + in(M_n)]}{A;\Gamma \vdash \mathsf{in}(M_1,\ldots,M_n).S : e}$$

OUTPUT
$$\frac{\forall i \in [1,n] \qquad \Gamma \vdash M_i : Un \qquad A;\Gamma \vdash S : e}{A;\Gamma \vdash \mathsf{out}(M_1,\ldots,M_n).S : e}$$

NEW NAME
$$\frac{A;\Gamma, n : Un \vdash S : e + [fresh(n)]}{A;\Gamma \vdash \mathsf{new}(n : Un).S : e}$$

NEW SECRET NONCE
$$\frac{A;\Gamma, n : nonce(I,A,M) \vdash S : e + [fresh(n,I,A,M)]}{A;\Gamma \vdash \mathsf{new}(n : nonce(I,A,M)).S : e}$$

SYMMETRIC ENCRYPT
$$\frac{A;\Gamma \vdash \{M_1,\ldots,M_n\}_K : enc(f) \qquad A;\Gamma, z : Un \vdash S : e}{A;\Gamma \vdash \mathsf{encrypt}\ \{M_1,\ldots,M_n\}_K\ \mathsf{as}\ z.S : e + [f]}$$

ASYMMETRIC ENCRYPT
$$\frac{A;\Gamma \vdash \{|M_1,\ldots,M_n|\}_K : enc(f) \qquad A;\Gamma, z : Un \vdash S : e}{A;\Gamma \vdash \mathsf{encrypt}\ \{|M_1,\ldots,M_n|\}_K\ \mathsf{as}\ z.S : e + [f]}$$

SYMMETRIC DECRYPT
$$\frac{\begin{array}{c}\Gamma \vdash M : Un \qquad fv(M_1,\ldots,M_n) = x_1,\ldots,x_m \qquad A;\Gamma, x_1 : T_1,\ldots,x_m : T_m \vdash S : e + [dec\{M_1,\ldots,M_n\}_K] \\ T_1,\ldots,T_m \text{ are the highest types such that } I;\Gamma_{\overline{A}}, x_1 : T_1,\ldots,x_m : T_m \vdash \{M_1,\ldots,M_n\}_K : enc(f)\end{array}}{A;\Gamma \vdash \mathsf{decrypt}\ M\ \mathsf{as}\ \{M_1,\ldots,M_n\}_K.S : e}$$

ASYMMETRIC DECRYPT
$$\frac{\begin{array}{c}\Gamma \vdash M : Un \qquad fv(M_1,\ldots,M_n) = x_1,\ldots,x_m \qquad A;\Gamma, x_1 : T_1,\ldots,x_m : T_m \vdash S : e + [dec\{|M_1,\ldots,M_n|\}_{\overline{K}}] \\ T_1,\ldots,T_m \text{ are the highest types such that } I;\Gamma_{\overline{A}}, x_1 : T_1,\ldots,x_m : T_m \vdash \{|M_1,\ldots,M_n|\}_{\overline{K}} : enc(f)\end{array}}{A;\Gamma \vdash \mathsf{decrypt}\ M\ \mathsf{as}\ \{|M_1,\ldots,M_n|\}_K.S : e}$$

sage M with J. That ciphertext is received by J which authenticates I and M by POSH COMMIT .

POSH REQUEST The judgment allows the encryption of $\{\mathsf{Id}(Id), C(N), \mathsf{Auth}(M), \widetilde{M}\}_K$. The tag of N, which is the nonce received from the network, and the identity label Id change according to the type of K. If K is a long-term key shared with J, then I may alternatively communicate to J that she is the claimant (tag Claim) of the authentication session or J is her intended verifier (tag Verif). Since K is only known by I and J, the two different tags above convey the same information: "I is the claimant and J is the intended verifier". In fact, communicating who is the claimant implicitly communicates who is the intended verifier and vice-versa. If K is the private key of I, then the intended verifier has to be specified, since the identity of the claimant is implicit in the signature. The message to be authenticated is tagged by Auth. Since the encryption represents the start of an authentication session from I to J, at least one occurrence of $run(I,J,M)$ has to be present in the typing effect. Notice that the judgment $\Gamma \vdash \widetilde{M} : Un$ says that M is a tuple of untagged terms. This makes the application of the rule deterministic.

POSH COMMIT After having received the above described ciphertext and checked the freshness of n, J accepts the authentication request from I and authenticates M. The check of n is formalized by removing the atomic effect $fresh(n)$, previously added by NEW NAME. This ensures that n is not checked again.

SOPH/SOSH Nonce Hand-Shake

J may start a *SOPH/SOSH* nonce-handshake with I for authenticating M, by creating a fresh nonce n with type $nonce(I,J,M)$ by NEW SECRET NONCE (Table 4). The nonce is then encrypted by SOPH/SOSH INQUIRY in a ciphertext and sent to I. The ciphertext is used for asking I whether or not she agrees on M and is willing to authenticate with J. When I receives the ciphertext, he may confirm the request by sending back the nonce either as cleartext or re-encrypted. In this case, the type of the nonce is cast to Un by SOPH/SOSH CONFIRM. When J receives that nonce, he authenticates I and M by SOPH/SOSH COMMIT .

SOPH/SOSH INQUIRY allows one entity to encrypt the secret nonce in messages of the form $\{\mathsf{Id}(Id), C(n), \mathsf{Auth}(M), \widetilde{M}\}_K$, where K is either the public key of I, or a long-term key shared with J. The tags used here are similar to POSH REQUEST . In this case, however, the tag of the nonce is *interrogative*, since the ciphertext is used for *asking I* whether or not she agrees on M and is willing to authenticate with J. This tagging disambiguates whether the ciphertext is used as challenge (as in *SOPH/SOSH*) or response (as in *POSH*).

SOPH/SOSH CONFIRM I may confirm her willingness to authenticate M with J by publishing the nonce a just received. As mentioned above, only terms with type Un can be sent as cleartext on the network: the rule casts the type of a from $nonce(I,J,M)$ to Un, while asserting the corresponding run[2].

[2]Notice that the rule is nondeterministic since the nonce is not indicated in the run primitive. If more than one nonce is present, one

Table 5 Authentication Rules

Encryption Rules

POSH REQUEST
$$\frac{\Gamma \vdash N, M, \widetilde{M} : Un \qquad \Gamma \vdash K : T \qquad T, C, Id \text{ as in } POSH(I,J)}{I;\Gamma \vdash \{\mathsf{Id}(Id), C(N), \mathsf{Auth}(M), \widetilde{M}\}_K : enc(run(I,J,M))}$$

POSH(I,J)

T	C	Id
$key_{sym}(I,J)$	Claim	I
$key_{sym}(I,J)$	Verif	J
$key_{priv}(I)$	Verif	J

SOPH/SOSH INQUIRY
$$\frac{\Gamma \vdash N : nonce(I,J,M) \qquad \Gamma \vdash M, \widetilde{M} : Un \qquad \Gamma \vdash K : T \qquad T, C, Id \text{ as in } SOPH/SOSH(I,J)}{J;\Gamma \vdash \{\mathsf{Id}(Id), C(N), \mathsf{Auth}(M), \widetilde{M}\}_K : enc(\oslash)}$$

SOPH/SOSH(I,J)

T	C	Id
$key_{sym}(I,J)$	Claim?	I
$key_{sym}(I,J)$	Verif?	J
$key_{pub}(I)$	Verif?	J

SYMMETRIC ENCRYPTION
$$\frac{\forall i \in [1,n] \qquad \Gamma \vdash M_i : Un \qquad \Gamma(K) \in \{Un, key_{sym}(I,J)\}}{I;\Gamma \vdash \{M_1, \ldots, M_n\}_K : enc(\oslash)}$$

ASYMMETRIC ENCRYPTION
$$\frac{\forall i \in [1,n] \qquad \Gamma \vdash M_i : Un \qquad \Gamma(K) \in \{Un, key_{priv}(I), key_{pub}(J)\}}{I;\Gamma \vdash \{|M_1, \ldots, M_n|\}_K : enc(\oslash)}$$

Secret Nonce Typing

SOPH/SOSH CONFIRM
$$\frac{I;\Gamma, a : Un, \Gamma' \vdash S : e}{I;\Gamma, a : nonce(I,J,M), \Gamma' \vdash \mathsf{run}(I,J,M).S : e}$$

Authentication

POSH COMMIT
$$\frac{J;\Gamma \vdash S : e \qquad \Gamma \vdash n, M, \widetilde{M} : Un \qquad \Gamma \vdash K : T \qquad dec\{\mathsf{Id}(Id), C(n), \mathsf{Auth}(M), \widetilde{M}\}_K \in e \qquad T, C, Id \text{ as in } POSH(I,J)}{J;\Gamma \vdash \mathsf{commit}(J,I,M).S : e + [fresh(n)]}$$

SOPH/SOSH COMMIT
$$\frac{J;\Gamma \vdash S : e \qquad \Gamma \vdash n : nonce(I,J,M) \qquad in(n) \in e \text{ or } dec\{n, \widetilde{M}\}_K \in e}{J;\Gamma \vdash \mathsf{commit}(J,I,M).S : e + [fresh(n,I,J,M)]}$$

SOPH/SOSH COMMIT When J receives back the nonce n, either as cleartext or as ciphertext, and checks its freshness, he authenticates I and M. As before, removing $fresh(n,I,J,M)$ from the final effect formalizes the linearity of nonce-check.

3.4 Typing Rules for Run-Time Behaviour

The typing rules in Table 6 are never used for the static analysis of a protocol. Instead, they are required in the proof that types are preserved at run-time.

TRUSTED and UNTRUSTED CIPHERTEXTS give the type Un to ciphertexts, allowing trusted principals to freely send and receive them. Notice that ciphertexts do not appear in the ρ-spi calculus syntax: they can be created either by the enemy or by trusted principals by means of encryption primitives. Ciphertexts created by a trusted principal I are regulated by the typing judgment $I;\Gamma \vdash \{M_1, \ldots, M_n\} : enc(f)$, while the ones created by the enemy are regulated by the ρ-spi calculus semantics (remember that each untagged term known by the environment has type Un). To motivate the sub-typing rule SECRET NONCE, let us consider the following protocol, using a *SOPH* nonce-handshake:

needs to "guess" which is going to be used first.

$$\begin{array}{llll} B & \rightarrow & A: & \{\mathsf{Id}(B), \mathsf{Auth}(m), \mathsf{Verif?}(n)\}_{\mathsf{Pub}(k_A)} \\ A & \rightarrow & B: & n \end{array}$$

Notice that A does not know whether the ciphertext has been created by B or by the enemy (remember: the enemy knows all the public keys). In the former case, the type of the nonce n is $nonce(A, B,M)$, in the latter it is Un. Hence, at run-time, the variable with type $nonce(A,B,M)$ might be substituted by a term with type Un. Moreover, the commit of B is typed by SOPH/SOSH COMMIT , which requires the nonce to have type $nonce(B,A,M)$. However, the type of n was cast by A to Un, in order to send back that nonce. The sub-typing rule SECRET NONCE addresses both of these issues.

Table 6 Run-Time Typing rules

The function $[M]$ is defined as follows: $\left\{ \begin{array}{lll} [M] & = & M \quad \text{if } M \text{ is either a name or a ciphertext or a key pair} \\ [C(M)] & = & [M] \quad \text{otherwise} \end{array} \right.$

We omit the asymmetric version of UNTRUSTED CIPHERTEXT and TRUSTED CIPHERTEXT.

TRUSTED CIPHERTEXT
$$\frac{I;\Gamma \vdash \{M_1,\ldots,M_n\}_{M_0} : enc(f)}{\Gamma \vdash \{M_1,\ldots,M_n\}_{M_0} : Un}$$

UNTRUSTED CIPHERTEXT
$$\frac{\forall i \in [0,n] \quad \Gamma \vdash [M_i] : Un}{\Gamma \vdash \{M_1,\ldots,M_n\}_{M_0} : Un}$$

SECRET NONCE
$$\frac{\Gamma \vdash M : Un}{\Gamma \vdash Un <: nonce(A,I,M)}$$

3.5 Safety Theorem

Our main result states that if a process can be typed with empty effect and empty typing environment, then every trace generated by that process is safe.

Theorem 1 (Safety) *If* $\oslash \vdash P : []$, *then P is safe.*

Interestingly, our analysis is strongly compositional, as stated by the following theorem. Let $\mathbf{keys}(k_1,\ldots,k_n)$ denote a sequence of key declarations.

Theorem 2 (Strong Compositionality) *Let P be the process* $\mathbf{keys}$ $(k_1,\ldots,k_n).(I_1 \triangleright !S_1 | \ldots | I_n \triangleright !S_n)$. *Then* $\oslash \vdash P : []$ *if and only if* $\oslash \vdash \mathbf{keys}(k_1,\ldots,k_n).I_i \triangleright !S_i : [], \forall i \in [1,n]$.

As a corollary, a protocol is safe if so are all the protocol participants. In addition, judging a participant safe only requires knowledge of the long-term keys it shares with other participants. This is a fairly mild assumption as the information conveyed by the keys is relative to identities of the parties sharing them, not to the protocol they are running. Consequently, unlike similar results proved of existing typing systems for authentication, notably [11, 12], Theorem 2 may be directly applied to the verification of multi-protocol systems (see [20, 21] for details). This flexibility has a price, however, in that our result relies critically on the run-time checks on the message tags provided by pattern-matching.

4. A CASE STUDY

We illustrate our system with a slightly modified version of the SPLICE/AS Protocol (see the remark at the end of this section):

Msg 1	B	$\rightarrow$	A:	B, n_B
Msg 2	A	$\rightarrow$	B:	$A, B, \{\|B, n_B, \{\|n_A\|\}_{\mathsf{Pub}(k_B)}\|\}_{\mathsf{Priv}(k_A)}$
Msg 3	B	$\rightarrow$	A:	$\{\|B, n_A\|\}_{\mathsf{Pub}(k_A)}$

This protocol should mutually authenticate A and B. In the first two messages, B exploits a POSH nonce handshake to authenticate A, as n_B is sent in clear and received encrypted. In the second message, A sends a (nested) challenge to B using a SOSH scheme. However, this second part is flawed as shown by the following attack sequence:

Msg 1.a	B	$\rightarrow$	E:	B, n_B
Msg 1.b	$E(B)$	$\rightarrow$	A:	B, n_B
Msg 2.b	A	$\rightarrow$	$E(B)$:	$A, B, \{\|B, n_B, \{\|n_A\|\}_{\mathsf{Pub}(k_B)}\|\}_{\mathsf{Priv}(k_A)}$
Msg 2.a	E	$\rightarrow$	B:	$E, B, \{\|B, n_B, \{\|n_A\|\}_{\mathsf{Pub}(k_B)}\|\}_{\mathsf{Priv}(k_E)}$
Msg 3.a	B	$\rightarrow$	E:	$\{\|B, n_A\|\}_{\mathsf{Pub}(k_E)}$
Msg 3.b	$E(B)$	$\rightarrow$	A:	$\{\|B, n_A\|\}_{\mathsf{Pub}(k_A)}$

There are two parallel sessions: a and b. Session a is between E and B, and E exploits such a session to impersonate B with A in session b. In particular, the enemy uses B as an oracle to decrypt message $\{\|n_A\|\}_{\mathsf{Pub}(k_B)}$ (messages 2.a and 3.a). Looking at correspondence assertions we have a $commit(A,B)$ [3] (session b) which does not match $run(B,E)$ (session a). Notice that this attack is similar to the well known one on the Needham-Schroeder public-key protocol [17]. In ρ-spi calculus , we capture these attack sequences thanks to the possibility of providing the enemy with long-term keys.

Since the protocol is unsafe it cannot be type-checked. We give the intuition of why this happens. A may be specified as follows:

(Msg 1)	$\mathsf{in}(B,x)$.
(Msg 2)	$\mathsf{new}(n_A)$. $\mathsf{encrypt}\ \{n_A\}_{\mathsf{Pub}(k_B)}$ as z . $\mathsf{encrypt}\ \{B,x,z\}_{\mathsf{Priv}(k_A)}$ as w . $\mathsf{out}(A,B,w)$.
(Msg 3)	$\mathsf{in}(y).\mathsf{decrypt}\ y$ as $\{B,n_A\}_{\mathsf{Priv}(k_A)}.\mathsf{commit}(A,B)$

There is no tagging that makes the sequential process above type-check. In particular, rule SOPH/SOSH COMMIT requires that the nonce n_A has type $nonce(B,A)$. This implies that the encryption of this nonce is regulated by the rule SOPH/SOSH INQUIRY . This rules always requires an identity label tagged by Id to be encrypted with the nonce, which is not the case in encrypt $\{n_A\}_{\mathsf{Pub}(k_B)}$ as z.

Since type-checking fails because of a missing identifier, it is natural to add it in order to try and repair the protocol. As a matter of fact changing message 2 into:

$$\text{Msg 2} \quad A \quad \rightarrow \quad B: \quad A, B, \{|B, n_B, \{|\mathbf{A}, n_A|\}_{\mathsf{Pub}(k_B)}|\}_{\mathsf{Priv}(k_A)}$$

we obtain the repaired protocol suggested by Gavin Lowe in [19]. To see how this protocol is checked using our type and effect system, consider the following tagged version of it:

Msg 1	B	$\rightarrow$	A:	B, n_B
Msg 2	A	$\rightarrow$	B:	$A, B, \{\|\mathsf{Id}(B), \mathsf{Verif}(n_B), \{\|\mathsf{Id}(A), \mathsf{Verif?}(n_A)\|\}_{\mathsf{Pub}(k_B)}\|\}_{\mathsf{Priv}(k_A)}$
Msg 3	B	$\rightarrow$	A:	$\{\|B, n_A\|\}_{\mathsf{Pub}(k_A)}$

In Message 2, A communicates to B that B is the verifier of the current authentication session and asks B whether or not he is willing to start an authentication session with her. Thus, we tag n_B by Verif and n_A by Verif?. In the second ciphertext, B sends back the nonce thus no tag is needed. The ρ-spi calculus specification of the protocol is reported in Table 7: notice that we analyze an unbounded number of sessions, where A and B play both the initiator and the responder role. To check mutual authentication, we alternatively decorate the protocol with two different pairs of correspondence primitives $\mathsf{run}_1(I,J), \mathsf{commit}_1(J,I)$ and $\mathsf{run}_2(I,J), \mathsf{commit}_2(J,I)$. Each of these pairs checks one direction of authentication, i.e., Initiator with respect to Responder and vice-versa. For the sake of readability we have written both

[3]When two parties do not want to authenticate any message, we use an empty message $\star$, which is omitted by the specification together with $\mathsf{Auth}(\star)$

the decorations together, but the typing is performed by considering only one of them at a time. The protocol can be typed (with respect to each of the decorations) with empty effect and environment, i.e., $\emptyset \vdash Protocol_S : []$. The rule used for typing each primitive is reported on the right side.

This specification can be easily extended to prove the protocol correct even with an unbounded number of entities (see [10] for details).

Remark 1. In the original version of the SPLICE/AS protocol, the format of Message 2 is

$$A, B, \{|A, n_B, \{|n_A|\}_{\mathsf{Pub}(k_B)}|\}_{\mathsf{Priv}(k_A)}$$

with A, rather than B, as the entity identifier of the top-level encrypted packet. With our present system, there is no tagging for this message enabling a successful validation of the protocol. We believe the problem can be circumvented, and the protocol verified, by enhancing the system to allow (messages encrypted by) public keys to be treated as entity identifiers, and to be tagged as such.

5. CONCLUSION AND RELATED WORK

We have proposed a type and effect system for authentication protocols. We have tested our analysis on several authentication protocols. Some results are reported in Table 8: in all cases, the analysis provide safety proofs for the correct versions of the protocols, while it consistently fails to validate the flawed versions (CCITT X.509 passes our analysis since it provides authentication even if it is affected by an attack on confidentiality). The main advantages of our proposal are

- scalability: since the authentication guarantees are local, safe sequential processes (possibly modeling different protocols) may be safely composed together;
- limited human effort: tagging is simple as it just requires to disambiguate the meaning of identifiers and nonces in some encrypted messages;
- simplicity: the type and effect system is simple but expressive enough to verify many existing protocols.

Moreover, our analysis can verify authentication in presence of enemies provided with long-term keys. This kind of analysis is not carried out in [11, 12].

The set of rules presented here and in [10] is general enough for analyzing many of the authentication protocols presented in literature. However we are now studying a method to "mechanically" extend the set of authentication rules, so that the safety result is preserved. The idea is to give meta-rules that dictate the shape of "good rules". We are also developing a tool for type checking and tag inference.

Related Work. Tagging is not a new idea and it is proposed and used for verification purposes in [3, 4, 11, 12, 16]. Typically, tagging amounts to add a different label to each encrypted protocol message, so that ciphertexts cannot be confused. Our tagging is less demanding, as we do not require that every message is unambiguously tagged since we tag only certain components. In particular, for protocols implemented with stronger tagging techniques, our tags can be safely removed without compromising the protocols' safety.

The Strand Spaces formalism [13, 14, 15, 22] is an interesting framework for studying authentication. There are interesting similarities between our analysis and the way the three kinds of nonce-handshakes are checked in Strand Spaces. It would be interesting to explore how our type system could be applied in such a framework, in order to provide mechanical proofs of safety.

The recent work by Bodei *et al.* on a control-flow analysis for message authentication in *Lysa* [6, 5] is also strongly related to our present approach. The motivations and goals, however, are different, since message authentication concerns the origin of a message while agreement provides guarantees about the presence in the current session of the claimant and its willingness to authenticate with the verifier.

Finally, we are currently formally comparing our type and effect system with the one by Gordon and Jeffrey [11, 12] and investigating a possible encoding of the former within the latter.

6. REFERENCES

[1] M. Abadi and C.Fournet. Private authentication. In *Proceedings of the 2002 Workshop on Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 27–40. Springer-Verlag, 2003.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[4] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proceedings of Foundations of Software Science and Computation Structures*, pages 136–152, 2003.

[5] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW'03)*, pages 126–140. IEEE Computer Society Press, June 2003.

[6] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of the Workshop on Issues on the Theory of Security (WITS'04)*, ENTCS. Elsevier, 2004.

[7] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of ICALP 01*, volume 2076, pages 667–681. LNCS 2076, Springer Verlag, 2001.

[8] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Logic in Computer Science*, pages 157–166, 1999.

[9] M. Bugliesi, R. Focardi, and M. Maffei. Principles for entity authentication. In *Proceedings of 5th International Conference Perspectives of System Informatics (PSI 2003)*, volume 2890 of *Lecture Notes in Computer Science*, pages 294–307. Springer-Verlag, July 2003.

[10] M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2004.

[11] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 145–159. IEEE Computer Society Press, June 2001.

[12] A. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 77–91. IEEE Computer Society Press, 24-26 June 2002.

[13] J.D. Guttman, F.J. Thayer, J.A. Carlson, J.C.Herzog, J.D. Ramsdell, and B.T. Sniffen. Trust management in strand spaces: a rely-guarantee method. In *Proceedings of European Symposium on Programming (ESOP 2004)*, volume 2986 of *Lecture Notes in Computer Science*, pages 325–339. Springer-Verlag, 2004.

[14] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)*, pages 24–34. IEEE Computer Society Press, July 2000.

[15] Joshua D. Guttman and F. Javier Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[16] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of 13th IEEE Computer*

Table 7 Amended SPLICE/AS Protocol in ρ-spi calculus

$Protocol_S \triangleq$
let $k_A = \mathsf{asym\text{-}key}(A)$.
let $k_B = \mathsf{asym\text{-}key}(B)$.
let $k_E = \mathsf{asym\text{-}key}(E)$.
$(\!|\ {}_{\forall I \in \{A,B\}}^{\ } \ {}_{\forall J \in \{A,B,E\}, J \neq I}$
$I \triangleright !Initiator_S(I, J, \mathsf{Priv}(k_I), \mathsf{Pub}(k_J))\ |$
$I \triangleright !Responder_S(I, J, \mathsf{Priv}(k_I), \mathsf{Pub}(k_J)))$

$Initiator_S(I, J, \mathsf{Priv}(k_I), \mathsf{Pub}(k_J)) \triangleq$	
new$(n_I : Un)$.	NEW NAME
out(I, n_I).	OUTPUT
in(J, I, y).	INPUT
decrypt y as $\{\!|\mathsf{Id}(I), \mathsf{Verif}(n_I), z'|\!\}_{\mathsf{Pub}(k_J)}$.	ASYMMETRIC DECRYPT
commit$_2(I, J)$.	POSH COMMIT
decrypt z' as $\{\!|\mathsf{Id}(J), \mathsf{Verif?}(x)|\!\}_{\mathsf{Priv}(k_I)}$.	ASYMMETRIC DECRYPT
run$_1(I, J)$.	SOPH/SOSH CONFIRM
encrypt $\{\!|I, x|\!\}_{\mathsf{Pub}(k_J)}$ as z''.	ASYMMETRIC ENCRYPT
out(z'')	OUTPUT

$Responder_S(I, J, \mathsf{Priv}(k_I), \mathsf{Pub}(k_J)) \triangleq$	
new$(n_I : nonce(J, I))$.	NEW SECRET NONCE
in(J, x).	INPUT
run$_2(I, J)$.	RUN
encrypt $\{\!|\mathsf{Id}(I), \mathsf{Verif?}(n_I)|\!\}_{\mathsf{Pub}(k_J)}$ as z.	SOPH/SOSH INQUIRY
encrypt $\{\!|\mathsf{Id}(J), \mathsf{Verif}(x), z|\!\}_{\mathsf{Priv}(k_I)}$ as z'.	POSH REQUEST
out(I, J, z').	OUTPUT
in(y).	INPUT
decrypt y as $\{J, n_I\}_{\mathsf{Priv}(k_I)}$.	ASYMMETRIC DECRYPT
commit$_1(I, J)$	SOPH/SOSH COMMIT

Table 8 Some Case Studies

Protocols	**Correct**	**Flawed**
CCITT X.509	X	
SPLICE/AS		X
Lowe's fixed version of SPLICE/AS	X	
Needham-Schroeder Public Key Protocol		X
Lowe's fixed version of Needham-Schroeder Public Key Protocol	X	
ISO's Symmetric Key Three Pass Mutual Authentication Protocol	X	
Private Authentication Protocols [1]	X	

Security Foundations Workshop (CSFW'00), pages 255–268. IEEE Computer Society Press, July 2000.

[17] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[18] G. Lowe. "A Hierarchy of Authentication Specification". In *Proceedings of the 10th Computer Security Foundation Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society Press, 1997.

[19] G. Lowe. "Towards a Completeness Result for Model Checking of Security Protocols". In *Proceedings Eleventh IEEE Computer Security Foundation Workshop, (CSFW'98)*. IEEE Computer Society Press, June 1998.

[20] M.Maffei. Tags for multi-protocol authentication. In *Proceedings of the 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems (SECCO '04). To appear*. ENTCS, August 2004.

[21] R.Focardi and M.Maffei. ρ-spi calculus at work: Authentication case studies. In *Proceedings of Mefisto Project. To appear*. ENTCS, March 2004.

[22] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. Journal of Computer Security, 1999. 7(2/3).

[23] Thomas Y. C. Woo and Simon S. Lam. A lesson on authentication protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[24] T.Y.C. Woo and S.S. Lam. "A Semantic Model for Authentication Protocols". In *Proceedings of 1993 IEEE Symposium on Security and Privacy*, pages 178–194, 1993.

APPENDIX

A. SEMANTICS OF ρ-spi

The dynamics of ρ-spi is formalized by means of a transition relation between *configurations*, i.e., pairs $\langle s, P\rangle$, where $s \in Act^*$ is a trace, P is a (closed) process. Each transition $\langle s, P\rangle \longrightarrow \langle s :: \alpha, P'\rangle$ simulates one computation step in P and records the corresponding action in the trace. The transitions involving a sequential process preserve the identity identifiers associated with the process, as in $\langle s, I \triangleright \pi.S\rangle \longrightarrow \langle s :: \alpha, I \triangleright S\rangle$, where α is the action corresponding to the primitive π. The set of all possible actions, noted Act, includes the action $out(M_1, \ldots, M_m)$ generated by output, $run(I,J,M)$ and $commit(I,J,M)$ by 'run' and 'commit', $in(M_1, \ldots, M_m)$ by input, $decrypt\{M_1, \ldots, M_m\}_K$ and $decrypt\{\!|M_1, \ldots, M_m|\!\}_{\mathsf{Key}(K)}$ by symmetric and asymmetric decryption, $encrypt\{M_1, \ldots, M_m\}_K$ and $encrypt\ \{M_1, \ldots, M_m\}_{\mathsf{Key}(K)}$ by symmetric and asymmetric encryption, $fresh(n)$ by restriction, $sym\text{-}key(k, I_1, I_2)$ and $asym\text{-}key(k, I)$ by symmetric and asymmetric key assignment.

The transitions, in Table 10, are mostly standard. PAR, REPLI-

Table 9 Most General Unifier

(Most General Unifier)

$$
\begin{array}{l}
m.g.u.(C(M_1),C(M_2)) = m.g.u.(M_1,M_2)\\
m.g.u.(n,n) = []\\
m.g.u.(n,x) = [n/x]\\
m.g.u.(\mathsf{Key}(M),\mathsf{Key}(M)) = []\\
m.g.u.(\mathsf{Key}(M),x) = [\mathsf{Key}(M)/x]\\
m.g.u.(\{M_1,\ldots,M_n\}_K,x) = [\{M_1,\ldots,M_n\}_K/x]\\
m.g.u.(\{|M_1,\ldots,M_n|\}_{\mathsf{Key}(K)},x) = [\{|M_1,\ldots,M_n|\}_{\mathsf{Key}(K)}/x]\\
m.g.u.((M_1,M_2,\ldots,M_n),(M'_1,M'_2,\ldots,M'_n)) = U \circ m.g.u.(U(M_2,\ldots,M_n),U(M'_2,\ldots,M'_n)),\\
\quad \text{where } U = m.g.u.(M_1,M'_1)
\end{array}
$$

CATION, OUTPUT, RUN and COMMIT are self-explanatory. Rule INPUT requires messages read from the network to be computable by the environment: the environment knowledge is defined by the message manipulation rules (see below). Moreover, input messages can be tested by pattern-matching, a capability that is also available upon decryption (cf. SYMMETRIC and ASYMMETRIC DECRYPT). Notice that, in ASYMMETRIC DECRYPT, a message encrypted with a public key can be decrypted only by the corresponding private key and vice-versa. It is worth noting that in ρ-spi calculus signature and encryption are the inverse function of each other. Both SYMMETRIC and ASYMMETRIC ENCRYPT behave as expected. Finally, restrictions of both names and keys are simply formalized as semantic transitions.

Pattern-matching is formalized by the notion of most general unifier, defined in Table 9. Notably, we prevent variables to be substituted by tagged terms, since tags are crucial patterns for authentication and we require them to be explicitly indicated. Also key-pairs cannot be bound to variables as to prevent private keys to be deduced by the corresponding public key.

We use a number of notation conventions. The restriction operator $\mathsf{new}(n).S$ is a binder for name n, the key declarations let $k = \mathsf{sym\text{-}key}(I_1,I_2)$ and let $k = \mathsf{asym\text{-}key}(I)$ are binders for k, while the input and decryption primitives are binders for the variables that occur in components M_i; finally, encryption is a binder for variable x. In all cases the scope of the binders is the continuation process. The notions of free/bound names and variables arise as expected. As in companion transition systems, see, e.g. [8], we identify processes up to renaming of bound variables and names, i.e., up to α-equivalence. Moreover we assume two infinite sets of bound names and free names so that bound names are distinct from free names and not touched by substitutions.

The message manipulation rules, in Table 11, formalize the environment actions. Rule OUT says that every message sent on the network is known by the environment. ENV allows the environment to generate a new bound name, not occurring in the trace. TAG and UNTAG allow the environment to tag and untag messages. By KEY PAIR, given a term M, the environment can build the private and public component of M. By SYMMETRIC ENCRYPTION and ASYMMETRIC DECRYPTION, if the environment knows $M_0,M_1,\ldots,M_n$, then it can encrypt $M_1,\ldots,M_n$ with M_0. SYMMETRIC DECRYPTION and ASYMMETRIC DECRYPTION formalize the capability of the environment to decrypt ciphertexts, whose decryption key is known. By PUBLIC KEYS, all the public keys are known by the environment. Moreover, by ENEMY KEYS, the environment may be provided with own key pairs and with long-term keys shared among the other participants. This gives the possibility to the enemy to start authentication sessions and, generally speaking, to interact with the other participants by pretending to be a trusted principal.

As an example, let us consider the following transitions :

$$
\begin{array}{l}
\langle s, A \triangleright \mathsf{in}(y).\mathsf{decrypt}\ y\ \mathsf{as}\ \{|n|\}_{\mathsf{Priv}(k_A)}.\mathbf{0}\rangle \ \rightarrow\\
\langle s :: in(\{|n|\}_{\mathsf{Pub}(k_A)}), A \triangleright \mathsf{decrypt}\ \{|n|\}_{\mathsf{Pub}(k_A)}\ \mathsf{as}\ \{|n|\}_{\mathsf{Priv}(k_A)}.\mathbf{0}\rangle \rightarrow\\
\langle s :: in(\{|n|\}_{\mathsf{Pub}(k_A)}) :: decrypt\{|n|\}_{\mathsf{Pub}(k_A)}, \mathbf{0}\rangle
\end{array}
$$

where $asym-key(k_A,A) \in s$. A decrypts the ciphertext by her own private key $\mathsf{Priv}(k_A)$. The operation is successful if the ciphertext is encrypted with the corresponding public-key. The must general unifier for input is $[\{|n|\}_{\mathsf{Pub}(k_A)}/y]$.

Table 10 Transition System for ρ-spi

Transition rules: bn(s) denotes the bound names in s. We omit the symmetric rule of PAR.
ciphertext. M ranges over terms: $M ::= m, x, \{M_1,\ldots,M_n\}_{M_0}, \mathsf{key}(M), C(M)$

PAR
$$\frac{\langle s,P\rangle \;\rightarrow\; \langle s',P'\rangle}{\langle s,P|Q\rangle \rightarrow \langle s',P'|Q\rangle}$$

REPLICATION
$$\langle s, I\triangleright !S\rangle \rightarrow \langle s, I\triangleright S|I\triangleright !S\rangle$$

OUTPUT
$$\langle s, I\triangleright \mathsf{out}(M_1,\ldots,M_n).S\rangle \;\rightarrow\; \langle s :: out(M_1,\ldots,M_n), I\triangleright S\rangle$$

RUN
$$\langle s, I\triangleright \mathsf{run}(I,J,M).S\rangle \;\rightarrow\; \langle s :: run(I,J,M), I\triangleright S\rangle$$

COMMIT
$$\langle s, I\triangleright \mathsf{commit}(I,J,M).S\rangle \;\rightarrow\; \langle s :: commit(I,J,M), I\triangleright S\rangle$$

INPUT
$$\frac{\forall i\in[1,n] \qquad s\vdash M_i' \qquad \exists U = m.g.u.((M_1',\ldots,M_n'),(M_1,\ldots,M_n))}{\langle s, I\triangleright \mathsf{in}(M_1,\ldots,M_n).S\rangle \;\rightarrow\; \langle s :: in(M_1',\ldots,M_n'), I\triangleright U(S)\rangle}$$

SYMMETRIC DECRYPT
$$\frac{\exists U = m.g.u.((M_1',\ldots,M_n'),(M_1,\ldots,M_n))}{\langle s, I\triangleright \mathsf{decrypt}\ \{M_1',\ldots,M_n'\}_K\ \mathsf{as}\ \{M_1,\ldots,M_n\}_K.S\rangle \;\rightarrow\; \langle s :: decrypt\{M_1',\ldots,M_n'\}_K, I\triangleright U(S)\rangle}$$

ASYMMETRIC DECRYPT
$$\frac{\exists U = m.g.u.((M_1',\ldots,M_n'),(M_1,\ldots,M_n))}{\langle s, I\triangleright \mathsf{decrypt}\ \{|M_1',\ldots,M_n'|\}_{\mathsf{Key}(K)}\ \mathsf{as}\ \{|M_1,\ldots,M_n|\}_{\overline{\mathsf{Key}}(K)}.S\rangle \;\rightarrow\; \langle s :: decrypt\{|M_1',\ldots,M_n'|\}_{\mathsf{Key}(K)}, I\triangleright U(S)\rangle}$$

SYMMETRIC ENCRYPT
$$\langle s, I\triangleright \mathsf{encrypt}\ \{M_1,\ldots,M_n\}_K\ \mathsf{as}\ x.S\rangle \;\rightarrow\; \langle s :: encrypt\{M_1,\ldots,M_n\}_K, I\triangleright S[\{M_1,\ldots,M_n\}_K/x]\rangle$$

ASYMMETRIC ENCRYPT
$$\langle s, I\triangleright \mathsf{encrypt}\ \{|M_1,\ldots,M_n|\}_{\mathsf{Key}(K)}\ \mathsf{as}\ x.S\rangle \;\rightarrow\; \langle s :: encrypt\{|M_1,\ldots,M_n|\}_{\mathsf{Key}(K)}, I\triangleright S[\{|M_1,\ldots,M_n|\}_{\mathsf{Key}(K)}/x]\rangle$$

RES
$$\frac{n\notin \mathrm{bn}(s)}{\langle s, I\triangleright \mathsf{new}(n).S\rangle \rightarrow \langle s :: fresh(n), I\triangleright S\rangle}$$

SIMMETRIC KEY
$$\frac{k\notin \mathrm{bn}(s)}{\langle s, \mathsf{let}\ k = \mathsf{sym\text{-}key}(I_1,I_2).P\rangle \rightarrow \langle s :: sym-key(k,I_1,I_2), P\rangle}$$

ASYMMETRIC KEY
$$\frac{k\notin \mathrm{bn}(s)}{\langle s, \mathsf{let}\ k = \mathsf{asym\text{-}key}(I).P\rangle \rightarrow \langle s :: asym-key(k,I), P\rangle}$$

Table 11 Message Manipulation Rules

AX
$$\frac{out(\ldots,M,\ldots)\in s}{s\vdash M}$$

ENV
$$\frac{n\notin \mathrm{bn}(s)}{s\vdash n}$$

TAG
$$\frac{s\vdash M}{s\vdash C(M)}$$

UNTAG
$$\frac{s\vdash C(M)}{s\vdash M}$$

KEY PAIR
$$\frac{s\vdash n}{s\vdash \mathsf{Pub}(n), \mathsf{Priv}(n)}$$

SYMMETRIC ENCRYPTION
$$\frac{\forall i\in[0,n] \quad s\vdash M_i}{s\vdash \{M_1,\ldots,M_n\}_{M_0}}$$

ASYMMETRIC ENCRYPTION
$$\frac{\forall i\in[0,n] \quad s\vdash M_i}{s\vdash \{|M_1,\ldots,M_n|\}_{M_0}}$$

SYMMETRIC DECRYPTION
$$\frac{s\vdash \{M_1,\ldots,M_n\}_{M_0} \qquad s\vdash M_0}{s\vdash M_i \qquad i\in[1,n]}$$

ASYMMETRIC DECRYPTION
$$\frac{s\vdash \{|M_1,\ldots,M_n|\}_{\mathsf{key}(M_0)} \qquad s\vdash \overline{\mathsf{key}}(M_0)}{s\vdash M_i \qquad i\in[1,n]}$$

PUBLIC KEYS
$$\frac{asym-key(k,I)\in s}{s\vdash \mathsf{Pub}(k)}$$

ENEMY KEYS
$$\frac{asym-key(k,E)\in s \vee sym-key(k,E,I)\in s}{s\vdash k}$$