On SDSI's Linked Local Name Spaces

Martín Abadi
Digital Equipment Corporation
Systems Research Center
ma@pa.dec.com

Abstract

Rivest and Lampson have recently introduced SDSI, a Simple Distributed Security Infrastructure. One of the important innovations of SDSI is the use of linked local name spaces. This paper suggests a logical explanation of SDSI's local name spaces, as a complement to the operational explanation given in the SDSI definition.

1. Linked Local Names Spaces

Rivest and Lampson have recently introduced a Simple Distributed Security Infrastructure (SDSI) [9]. One of the important innovations of SDSI is the use of linked local name spaces. In SDSI, each principal has a name space where names are bound to values, possibly by reference to the name spaces of other principals.

For example, in a particular local name space, the name `Lampson` may be associated with a public key KL. As a consequence of this binding, any statement whose signature can be verified using KL will be viewed as coming from `Lampson`. In addition, the name `Rivest` may be associated with `Lampson's Rivest`. As a consequence of this binding, if `Lampson` says that `Rivest`'s public key is KR, then any statement whose signature can be verified using KR will be viewed as coming from `Rivest`. Compound names like `Lampson's Rivest` allow one name space to import bindings from another.

Linked local name spaces offer the promise of combining some of the advantages of PGP-style local certification with those of hierarchical certification schemes (see, e.g., [3, 5, 11, 2, 4]). In particular, SDSI certification is egalitarian, and does not need to assume any global trust or any global notion of identity (beyond that inherent in public-key cryptography). On the other hand, SDSI can take advantage of structured trust relations and naming conventions, when such exist.

The precise rules for local names are not as simple as our first example may have suggested. Those rules deal with complex names such as `Lampson's Rivest's secretary` and `Lampson's Rivest's students`, and also with DNS names (that is, Internet e-mail names). The SDSI definition gives a rather operational account of local names. Basically, it explains local names by giving an algorithm for name resolution that maps names to their meanings.

This paper suggests an alternative account of local names. This account is based on a logic where one can express compound names such as `Lampson's Rivest`, and assert that `Lampson's Rivest` is bound to the public key KR. One can also ask whether `Lampson's Lampson` is `Lampson`, and whether `Lampson's Rivest's secretary` is `Lampson's (Rivest's secretary)`. These questions can be addressed independently of the binding for `Lampson`, and even when name resolution is not possible because `Lampson` has not yet been bound.

The purpose of the logic is to explain local names in a general, self-contained way, without requiring reference to particular implementations. Nevertheless, the SDSI name-resolution algorithm can be recast as a sound proof method within the logic. Hopefully, this work will contribute to the understanding of naming in SDSI and in related systems such as Simple Public Key Infrastructure [4].

The next section introduces basic concepts and notations. Section 3 develops a logic for linked local name spaces, and proves the soundness of the SDSI name-resolution algorithm with respect to the logic. It also contains an example. Section 4 defines a semantics for the logic. The semantics serves as a mathematical tool, and as a precise counterpart to some of the informal explanations of naming. Section 5 concerns self-reference. An appendix contains all proofs.

This work can be seen as a descendant of the work of Lampson et al. in [8, 1, 10]. That work develops a calculus of principals and their statements, analogous to that of section 3; however, it does not explore the SDSI approach to naming.

At this time (March 1997), a precise specification document for SDSI is not yet available. All the information about SDSI contained here comes from a comprehensive but informal description [9]. SDSI is a rather sophisticated design, with many aspects. This paper largely ignores aspects of SDSI other than linked local name spaces.

2. Concepts and Notations

This section is an informal discussion of basic concepts related to linked local name spaces. The main goal of this section is to review some of the ideas of SDSI at a suitable level of abstraction and to clarify their meanings. This section also introduces notations used in the rest of the paper.

2.1. Keys and Names

Hierarchical certification schemes rely on a general agreement about the roots of certification hierarchies; the names of these roots should be interpreted similarly in all name spaces. On the other hand, in egalitarian certification schemes, each principal can introduce names of its own, for private use. SDSI supports both styles of naming. In our study of SDSI, we therefore deal with both global and local ways of referring to principals:

- *Global identifiers* are understood equally in all name spaces. Global identifiers include public keys and global names. Public keys are global because any two principals should agree on the value of a public key when they see it, and should agree that a statement is signed using the inverse of a public key when they see the public key and the statement. In addition, the names of certain "special roots" are global, by special dispensation from the designers of SDSI; for example, `DNS`!! is a global name, and represents the root of the DNS hierarchy.
- *Local names* are the auxiliary names that each principal can use in its name space. For example, `Lampson` may be used as a local name by anyone, and bound to any value. The value could be a global identifier (such as a public key), or another local name. Since two principals may disagree on the value of `Lampson`, they may not agree on whether a statement comes from `Lampson`.

We assume a set of global identifiers G and a disjoint set of local names N. The symbols f, g, and h range over G, and the symbols m and n over N.

2.2. Compound Names

In SDSI, the use of compound names is the basic mechanism for linking local name spaces. A compound name is an expression of the form $(\texttt{ref:}\, p_1 \ldots p_n)$, which is abbreviated to $p_1\texttt{'s} \ldots \texttt{'s}\, p_n$. Each of $p_1, \ldots, p_n$ may be a global identifier, a local name, or a compound name. Intuitively, $p\texttt{'s}\, q$ is the principal that is called q in the name space of the principal that is called p in the current name space. In general, $p_1\texttt{'s} \ldots \texttt{'s}\, p_n$ is the principal that is called p_n in the name space …of the principal that is called p_1 in the current name space.

Going beyond these first intuitions about compound names, some questions immediately arise:

- Does $(\texttt{ref:}\, p_1 \ldots p_n)$ make sense when $n = 0$? SDSI seems to allow this case. The meaning of $(\texttt{ref:}\,)$ is the current principal—whoever is trying to do the name resolution.
- Does $(\texttt{ref:}\, p_1 \ldots p_n)$ make sense when $n = 1$? SDSI explicitly allows this case, and says that $(\texttt{ref:}\, p)$ is equivalent to p, since the first argument to `ref` is always interpreted in the current name space.
- Is $((p\texttt{'s}\, q)\texttt{'s}\, r)$ the same as $(p\texttt{'s}\, q\texttt{'s}\, r)$ and the same as $(p\texttt{'s}\, (q\texttt{'s}\, r))$? This is not asserted in the SDSI paper, but is consistent with it. It is valid in the logic of this paper.
- Is $(p\texttt{'s}\, p)$ the same as p? SDSI does not assume this, as it would be unreasonable, for example in the case where p is `spouse`.

In light of these properties, we consider `ref` only in the binary case, without loss of generality:

- To recover the expressiveness of the nullary case ($n = 0$), we introduce the construct `Self`, and replace $(\texttt{ref:}\,)$ with `Self`. We postpone the study of `Self` to section 5.
- In the unary case, instead of writing $(\texttt{ref:}\, p)$, we can simply write p.
- The case where $n > 2$ is reducible to the binary case, since for example we can regard $(p\texttt{'s}\, q\texttt{'s}\, r)$ as an abbreviation for $((p\texttt{'s}\, q)\texttt{'s}\, r)$.

The set of *principal expressions* is the least set P containing G and N, and such that if $p, q \in P$ then $(p\texttt{'s}\, q) \in P$. Thus, a principal expression is a syntactic representation for a principal. The symbols p, q, and r range over P.

2.3. Statements

SDSI is concerned with statements that principals make. Those statements can be certificates and requests for service, in particular. All statements are understood with respect to a current name space.

A true statement may not be true forever. Consequently, certificates contain temporal information, indicating their periods of validity. For simplicity, we prefer to abstract away issues of time (for example, interesting questions about early versus late binding). Therefore, all statements are understood with respect to an arbitrary but fixed time, the current time.

We write p *says* s to indicate that the principal denoted by p makes statement s. When p is a public key K, the statement can be made by signing a textual representation of s using the private key that corresponds to K.

2.4. Bindings and Their Meanings

A *name space* is a set of bindings of names to values. We use the notation $n \mapsto v$ to represent the binding of the name n to the value v. The name n and the value v form a *name/value pair*. The value can be an arbitrary object, but the only case of relevance to this paper is that where the value is a principal expression. In a name space where name n is bound to principal expression p, any statement by p is taken as a statement by n.

Binding n to p is much like defining n to be p within the current name space. One may therefore view a binding as an assertion of equality. For example, one may bind the local name `lawyer` to the principal expression `Ted's lawyer` when `lawyer` and `Ted's lawyer` represent the same "real" principal.

Interpreting binding as equality is appealingly simple; but this interpretation runs into some trouble. Suppose, for example, that two different public keys $KL1$ and $KL2$ are values for the local name `Lampson` in some name space (perhaps because the "real" Lampson controls two keys). SDSI allows this:

> A local name can be undefined, or it can be bound to some value [...]. The principal may assign a value to a local name by issuing a corresponding certificate. If the local name already has a valid name/value certificate, the new certificate augments the old one, in the sense that a SDSI application is deemed to act correctly if it uses the name/value binding given in either certificate.

However, it is a little hard to make sense of the equalities $KL1 = \texttt{Lampson} = KL2$, since $KL1$ and $KL2$ are different. These equalities may make sense if we view $KL1$, `Lampson`, $KL2$ as being three ways of referring to one "real" principal. It is not clear whether this is an appropriate semantic intuition. One of the principles of SDSI is that "principals are public keys"; the equalities $KL1 = \texttt{Lampson} = KL2$ put us in an interesting situation where one principal is two different keys. Clearly, in this situation, the principle that "principals are public keys" should be understood loosely. Section 3 gives a stronger, formal argument against the equality interpretation.

In light of these difficulties, we may consider a weaker interpretation of binding. This weaker interpretation is possible because bindings need not be used in a symmetric way. For example, the binding of `lawyer` to `Ted's lawyer` may be used to turn a statement by `Ted's lawyer` into a statement by `lawyer`, not vice versa.

According to the weaker interpretation, a binding is simply an assertion of a *speaks-for* relation [8, 1]. Whenever p speaks for q, if p makes a statement then q makes the same statement. However, unlike equality, the speaks-for relation is not symmetric. With this weaker interpretation, the binding of `lawyer` to `Ted's lawyer` means that `Ted's lawyer` speaks for `lawyer`, not that `lawyer` is `Ted's lawyer`. Similarly, the binding of `Lampson` to $KL1$ and $KL2$ means that $KL1$ and $KL2$ both speak for `Lampson`, but does not imply the equality of `Lampson`, $KL1$, and $KL2$.

This paper generally favors the speaks-for interpretation of binding. However, it is not strictly necessary to commit to one specific interpretation. It suffices to assume properties of binding that seem compatible with either interpretation.

2.5. A General Form for Bindings

Suppose that a name space o contains the binding

$$\texttt{sysAdmin} \mapsto K1$$

and many bindings of the form

$$n \mapsto (\texttt{sysAdmin's CertAuthority's}\ n)$$

Suppose further that $K1$ says

$$\texttt{CertAuthority} \mapsto K2$$

Within the name space o, we may deduce that `sysAdmin's CertAuthority` corresponds to $K2$, and then we may use that fact to deduce

$$n \mapsto (K2'\texttt{s}\ n)$$

In this situation, it is useful to generalize the $\mapsto$ relation so that we can write $p \mapsto q$ even when p is not a simple name. For example, we may write

$$(\texttt{sysAdmin's CertAuthority}) \mapsto K2$$

in order to capture the association between the compound name `sysAdmin's CertAuthority` and the key $K2$. If name resolution was implemented using a cache, then the cache might record facts like

$$(\texttt{sysAdmin's CertAuthority}) \mapsto K2$$

In general, $p \mapsto q$ means that p is bound to q in the current name space, either as the result of an explicit binding (like $\texttt{sysAdmin} \mapsto K1$), or as a consequence of other bindings (like $(\texttt{sysAdmin's CertAuthority}) \mapsto K2$).

Reflexivity:	$p \mapsto p$
Transitivity:	$(p \mapsto q) \supset ((q \mapsto r) \supset (p \mapsto r))$
Left-monotonicity:	$(p \mapsto q) \supset ((p'\mathsf{s}\ r) \mapsto (q'\mathsf{s}\ r))$
Globality:	$(p'\mathsf{s}\ g) \mapsto g$ if g is a global identifier.
Associativity:	$((p'\mathsf{s}\ q)'\mathsf{s}\ r) \mapsto (p'\mathsf{s}\ (q'\mathsf{s}\ r))$ $(p'\mathsf{s}\ (q'\mathsf{s}\ r)) \mapsto ((p'\mathsf{s}\ q)'\mathsf{s}\ r)$
Linking:	$(p\ says\ (n \mapsto r)) \supset ((p'\mathsf{s}\ n) \mapsto (p'\mathsf{s}\ r))$ if n is a local name.
Speaking-for:	$(p \mapsto q) \supset ((q\ says\ s) \supset (p\ says\ s))$

Figure 1. Axioms for linked local name spaces.

3. A Logic for Linked Local Name Spaces

In this section, we define a logic based on the concepts and notations of section 2. We also explain the SDSI name-resolution algorithm as a particular proof method within the logic, and show that it is sound.

3.1. Syntax

First we introduce a syntax for statements, building on section 2. We assume a set S of atomic statements (written s_0, s_1, s_2, $\ldots$). The set of *formulas* (or *statements*) is the least set such that:

- an atomic statement is a formula;
- if s and t are formulas then so are $\neg s$ and $s \wedge t$; we write the implication $s \supset t$ as an abbreviation for $\neg(s \wedge \neg t)$;
- if p and q are principal expressions then $p \mapsto q$ is a formula;
- if p is a principal expression and s is a formula then $p\ says\ s$ is a formula.

The symbols s and t range over the set of formulas.

The intuitive meanings of principal expressions, of $\mapsto$, and of $says$ are those explained in section 2. As discussed there, all formulas are understood with respect to a current name space and a current time.

For example, $KL\ says\ (\texttt{Rivest} \mapsto KR)$ is a formula if KL and KR are global identifiers and `Rivest` is a local name. This formula expresses an assertion signed with the inverse of the public key KL. The assertion is that `Rivest` is bound to KR. This binding is asserted for the local name space associated with KL, not universally.

3.2. Axioms

Next we give a proof system; this proof system has three components:

1. The standard axioms and rules of propositional logic, such as the axiom $(s \wedge t) \supset s$ and the rule

$$\frac{s \quad s \supset t}{t}$$

2. The standard axiom and rule of modal logic (e.g., [6]):

$$(p\ says\ (s \supset t)) \supset ((p\ says\ s) \supset (p\ says\ t))$$

$$\frac{s}{p\ says\ s}$$

   These are useful for manipulating statements within the scope of $says$, but we do not actually need them in this paper. The axiom expresses that the statements that a principal says are closed under consequence; the rule expresses that every principal says all provable statements.

3. New axioms that deal with linked local name spaces, listed in Figure 1.

   Reflexivity and Transitivity express that the binding relation is reflexive and transitive.

   Left-monotonicity means that if p is bound to q then $p'\mathsf{s}\ r$ is bound to $q'\mathsf{s}\ r$. We do not have that $r'\mathsf{s}\ p$ is bound to $r'\mathsf{s}\ q$, because p may not be bound to q in r's name space.

   Globality expresses that, if g is a global identifier and p an arbitrary principal expression, then $p'\mathsf{s}\ g$ is bound to g. (Nevertheless, $p'\mathsf{s}\ g$ might be bound to other values as well.) This axiom is justified by the globality of g.

Associativity postulates that references are associative. Two formulas are used because we do not assume that $\mapsto$ is symmetric. (We actually need only the second formula.)

Linking expresses that if p says that n is bound to r (implicitly, in p's name space), then $p'\mathtt{s}\ n$ is bound to $p'\mathtt{s}\ r$ (implicitly, in the current name space). Thus, p is trusted on assertions about its name space.

Finally, Speaking-for expresses that if p is bound to q, then any statement by q is taken as a statement by p.

This axiom system could be extended in several ways. In particular, we could add:

(a) the converse of Globality: $g \mapsto (p'\mathtt{s}\ g)$ for g a global identifier;

(b) a generalization of Linking: $(p\ \mathit{says}\ (q \mapsto r)) \supset ((p'\mathtt{s}\ q) \mapsto (p'\mathtt{s}\ r))$ for q an arbitrary principal expression.

Both of these extensions may seem reasonable. Interestingly, however, there is a fundamental conflict between them. For instance, suppose that we have the formula $KL\ \mathit{says}\ (\mathtt{DNS!!} \mapsto KL)$, where KL is a public key; this formula expresses that KL says that DNS!! is bound to KL, a suspicious but possible statement. By (b), we can derive $(KL'\mathtt{s}\ \mathtt{DNS!!}) \mapsto (KL'\mathtt{s}\ KL)$; by (a), we can conclude $\mathtt{DNS!!} \mapsto KL$. This is an undesirable consequence: it means that a strange statement by an arbitrary key KL can pollute the current name space. By Speaking-for, once we have the binding $\mathtt{DNS!!} \mapsto KL$ any statement from KL will be viewed as a statement from DNS!!; thus, KL can hijack DNS!!. Therefore, (a) and (b) are incompatible.

Another possible addition is the following symmetry axiom:

(c) $(p \mapsto q) \supset (q \mapsto p)$.

This axiom is motivated by the equality interpretation of binding. Interestingly, it has undesirable consequences too. For instance, suppose that we have the formulas $KL\ \mathit{says}\ (n \mapsto KL)$ and $KL\ \mathit{says}\ (n \mapsto \mathtt{DNS!!})$, where n is a local name. By Linking, Globality, and Transitivity, we obtain $(KL'\mathtt{s}\ n) \mapsto KL$ and $(KL'\mathtt{s}\ n) \mapsto \mathtt{DNS!!}$. By the new symmetry axiom and Transitivity, we can conclude $\mathtt{DNS!!} \mapsto KL$. Thus, again, statements by an arbitrary key KL can pollute the current name space. This provides a strong argument against the adoption of symmetry, and (at least in the setting of our logic) against the equality interpretation of binding.

In the arguments about (a), (b), and (c), somewhat plausible axioms lead to catastrophic results. A bit of paranoia is therefore in order. How can we know that our present axiom system does not lead to exactly the same results? We prove that it does not in Corollary 6 of section 4. We show that, no matter what KL says, one cannot derive $\mathtt{DNS!!} \mapsto KL$. This proof should not be construed as a final guarantee of the security of our logic or of SDSI. It is however a useful sanity check.

3.3. An Example

In order to illustrate how the logic is used, we work through an example. Applying a SDSI abbreviation, we write `smith@aol.com` as a shorthand for

$$\mathtt{DNS!!'s\ com's\ aol's\ smith}$$

Note that we have omitted some parentheses in the expression $\mathtt{DNS!!'s\ com's\ aol's\ smith}$. By axiom Associativity, fortunately, we do not have to worry about how this expression is parenthesized. We omit parentheses in the rest of the example, with the same justification.

Consider the following bindings for the current name space and the following certificates:

1. the local name `BrokersInc` is bound to the public key $K1$,
2. the local name `broker` is bound to $\mathtt{BrokersInc's\ NYoffice's\ Smith}$,
3. a certificate signed with the inverse of $K1$ says that the local name `NYoffice` is bound to the public key $K2$,
4. a certificate signed with the inverse of $K2$ says that the local name `Smith` is bound to `smith@aol.com`.

Logically, we represent these bindings and certificates with the formulas:

1. $\mathtt{BrokersInc} \mapsto K1$,
2. $\mathtt{broker} \mapsto (\mathtt{BrokersInc's\ NYoffice's\ Smith})$,
3. $K1\ \mathit{says}\ (\mathtt{NYoffice} \mapsto K2)$,
4. $K2\ \mathit{says}\ (\mathtt{Smith} \mapsto \mathtt{smith@aol.com})$.

Using these formulas as hypotheses, we construct the semi-formal proof of Figure 2. Thus, we obtain that the local name `broker` is bound to `smith@aol.com`.

Following Lampson, we can examine a proof like this one much like we would examine an audit trail. We can identify which of the hypotheses are used, and how they contribute to the conclusion. In this case, it is clear, for example, that both of the certificates from $K1$ and $K2$ are relevant, and that $K1$ and $K2$ must be trusted to have issued them appropriately.

A name like `broker` may be used in an ACL. Given a signed request, one may need to determine whether

1. $\texttt{BrokersInc}$ *says* ($\texttt{NYoffice} \mapsto K2$) from hypotheses (1) and (3) by Speaking-for.

2. ($\texttt{BrokersInc}'\texttt{s NYoffice}$) $\mapsto$ ($\texttt{BrokersInc}'\texttt{s}\ K2$) from step (1) by Linking.

3. ($\texttt{BrokersInc}'\texttt{s NYoffice}$) $\mapsto K2$ from step (2) by Globality and Transitivity.

4. ($\texttt{BrokersInc}'\texttt{s NYoffice}$) *says* ($\texttt{Smith} \mapsto \texttt{smith@aol.com}$) from hypothesis (4) and step (3) by Speaking-for.

5. ($\texttt{BrokersInc}'\texttt{s NYoffice}'\texttt{s Smith}$) $\mapsto$ $\texttt{smith@aol.com}$ from step (4) by Linking, Globality, and Transitivity, since DNS!! is a global name and $\texttt{smith@aol.com}$ stands for DNS!!$'\texttt{s com}'\texttt{s aol}'\texttt{s smith}$.

6. $\texttt{broker} \mapsto \texttt{smith@aol.com}$ from hypothesis (2) and step (5) by Transitivity.

Figure 2. A proof.

that request comes from $\texttt{broker}$ in order to make an access-control decision. By Speaking-for and the binding $\texttt{broker} \mapsto \texttt{smith@aol.com}$, it suffices to determine whether the signature is that of $\texttt{smith@aol.com}$. This determination requires a chain of certificates starting from the root of the DNS hierarchy. However, the result $\texttt{broker} \mapsto \texttt{smith@aol.com}$ can be derived (and evaluated) before that chain of certificates is presented.

3.4. Simulating Name Resolution

Name resolution is the process of mapping a principal expression to a global identifier. Name resolution happens in the context of assumptions, in particular assumptions about the values of local names. The form of those assumptions may vary, depending on protocols and architecture. We consider a set of assumptions of the forms $n \mapsto q$ and g *says* $(n \mapsto q)$, where n is a local name, q a principal expression, and g a global identifier. The assumptions of the form $n \mapsto q$ correspond to bindings in the current name space. Those of the form g *says* $(n \mapsto q)$ correspond to certificates about bindings in other name spaces. Typically g is a public key K, and g *says* $(n \mapsto q)$ is implemented as a certificate signed with the inverse of K. We let E be the conjunction of all those assumptions. It is an easy exercise to include also bindings for global names (for example, a binding of DNS!! to a corresponding public key).

In SDSI, name resolution is specified as a recursive algorithm. The core of the algorithm is presented as a (nondeterministic) function $\texttt{REF2}(o, p)$ that returns the meaning of p in the name space associated with o. The argument o may be either $\texttt{current principal}$ ($\texttt{cp}$, for short) or a global identifier g.

Figure 3 gives an adaptation of SDSI's definition of REF2. It is fairly faithful, but simpler than the original because, for example, it does not deal with quoting or encrypted objects. We write assumptions(o) for the set of assumptions for o: (1) assumptions($\texttt{cp}$) consists of the assumptions of the form $n \mapsto q$, (2) assumptions(g) consists of the assumptions of the form g *says* $(n \mapsto q)$. The computation of $\texttt{REF2}(o, p)$ is nondeterministic because assumptions(o) may include several bindings of the form $n \mapsto q$ for a given n. We say that $\texttt{REF2}(o, p)$ yields f when f is one of the possible outputs of $\texttt{REF2}(o, p)$.

In this section, we show how REF2 can be simulated within our logic. We carry this out in two steps. First, we define a new set of rules, called the *name-resolution rules*. These rules rely on the notation of the logic, but constitute a separate proof system. We establish that the name-resolution rules are equivalent to REF2. As a second step, we show that the name-resolution rules are sound with respect to the logic.

The name-resolution rules are simple rules designed for proving formulas of the form $E \supset (p \mapsto f)$. They are given in Figure 4. One can understand this set of rules as an algorithm. The algorithm starts out with an input p and assumptions E; it finds a suitable f, if one exists. The algorithm does a case analysis on the form of p. There is one rule (Base case 1) for when p is a global identifier f, one rule (Local name 1) for when p is a local name n, and four rules for when p is of the form $q'\texttt{s}\ r$. Three of those four (Base case 2, Local name 2, Ref 2) do a case analysis on the form of r when q is a global identifier g. The remaining rule (Ref 1) applies whenever p is of the form $q'\texttt{s}\ r$.

Despite their apparent differences, the name-resolution rules and REF2 are equivalent, as the following proposition shows:

Proposition 1 *Assume that a set of bindings, each of the form $(n \mapsto q)$, is given for* $\texttt{cp}$ *and for each global identifier g. Let E be the conjunction of the formulas $(n \mapsto q)$, for each binding $(n \mapsto q)$ for* $\texttt{cp}$, *and g says $(n \mapsto q)$, for each binding $(n \mapsto q)$ for g. For every principal expression p and global identifiers g and f, we have:*

- *$E \supset (p \mapsto f)$ is provable with the name-resolution rules if and only if* $\texttt{REF2}(\texttt{cp}, p)$ *yields f.*

$$
\begin{array}{lll}
\mathtt{REF2}(o,p) & = & \text{if } p \text{ is a global identifier } f \\
 & & \text{then return } f \\
 & & \text{else if } p \text{ is a local name } n \\
 & & \quad\quad \text{and assumptions}(o) \text{ includes } n \mapsto q \\
 & & \text{then return } \mathtt{REF2}(o,q) \\
 & & \text{else if } p \text{ is a compound name } q'\mathtt{s}\ r \\
 & & \text{then return } \mathtt{REF2}(\mathtt{REF2}(o,q),r) \\
 & & \text{else fail}
\end{array}
$$

Figure 3. Name-resolution algorithm.

(Base case 1)

$$\frac{f \in G}{E \supset (f \mapsto f)}$$

(Base case 2)

$$\frac{f, g \in G}{E \supset ((g'\mathtt{s}\ f) \mapsto f)}$$

(Local name 1)

$$\frac{n \mapsto q \text{ is a conjunct in } E \qquad E \supset (q \mapsto f)}{E \supset (n \mapsto f)}$$

(Local name 2)

$$\frac{g\ \mathit{says}\ (n \mapsto q) \text{ is a conjunct in } E \qquad E \supset ((g'\mathtt{s}\ q) \mapsto f)}{E \supset ((g'\mathtt{s}\ n) \mapsto f)}$$

(Ref 1)

$$\frac{E \supset (q \mapsto f_0) \qquad E \supset ((f_0{}'\mathtt{s}\ r) \mapsto f)}{E \supset ((q'\mathtt{s}\ r) \mapsto f)}$$

(Ref 2)

$$\frac{g \in G \qquad E \supset ((g'\mathtt{s}\ q) \mapsto f_0) \qquad E \supset ((f_0{}'\mathtt{s}\ r) \mapsto f)}{E \supset ((g'\mathtt{s}\ (q'\mathtt{s}\ r)) \mapsto f)}$$

Figure 4. Name-resolution rules.

- *$E \supset ((g'\mathtt{s}\,p) \mapsto f)$ is provable with the name-resolution rules if and only if* $\mathtt{REF2}(g,p)$ *yields* f.

The next proposition relates the name-resolution rules to the logic of section 3.2:

Proposition 2 *The name-resolution rules are sound with respect to the logic. That is, if $E \supset (p \mapsto f)$ is provable using those rules, then it is also provable in the logic.*

We conclude:

Corollary 3 *The name-resolution algorithm is sound with respect to the logic. That is, given E as in Proposition 1 and any principal expression p, if* $\mathtt{REF2}(\mathtt{cp},p)$ *yields a global identifier f, then $E \supset (p \mapsto f)$ is provable in the logic.*

The converse of this property does not hold: sometimes $E \supset (p \mapsto f)$ is provable in the logic although $\mathtt{REF2}(\mathtt{cp},p)$ does not yield f. In particular, the logic's Globality axiom yields $(n'\mathtt{s}\,g) \mapsto g$ even when we do not know how n is defined. Furthermore, given the assumptions $m \mapsto f_1$, $m \mapsto f_2$, f_1 *says* $(n_1 \mapsto n_2)$, and f_2 *says* $(n_2 \mapsto h)$, the logic enables us to derive $(m'\mathtt{s}\,n_1) \mapsto h$, while $\mathtt{REF2}(\mathtt{cp}, m'\mathtt{s}\,n_1)$ does not yield h. It should be easy enough to restrict the logic in order to prevent these results, but it is not clear whether they are harmful, and they might in fact be useful.

The logic is more powerful than the algorithm also in that it permits more general arguments. For example, using the logic, one can reason that if $\mathtt{Rivest} \mapsto (\mathtt{Lampson'}\mathtt{s}\ \mathtt{Rivest})$ and $\mathtt{Lampson} \mapsto KL$ then $\mathtt{Rivest} \mapsto (KL'\mathtt{s}\ \mathtt{Rivest})$, independently of any certificates about $KL'\mathtt{s}\ \mathtt{Rivest}$. Without such certificates, in contrast, the algorithm fails.

4. Semantics

In this section, we define a semantics for the logic of section 3. By the standards of logicians, the semantics is rather elementary. It could probably be enhanced (and complicated) using ideas from the modal-logic literature (see e.g. [6, 1, 7]). Despite its simplicity, the semantics was helpful in refining the rules of section 3, and it can be rather useful, as we demonstrate with Corollary 6 below.

Recall that G is the set of global identifiers, N the set of local names, P the set of all principal expressions, and S the set of atomic propositions. The semantics assumes:

- a set $\mathcal{W}$,
- a mapping α from G to subsets of $\mathcal{W}$,
- a mapping ρ from $N \times \mathcal{W}$ to subsets of $\mathcal{W}$, and
- a mapping μ from $S \times \mathcal{W}$ to truth values.

In the vocabulary of modal logic, $\mathcal{W}$ is a set of possible worlds. The choice of the set $\mathcal{W}$ does not matter for our purposes. What matters is that $\mathcal{W}$ has a subset $\alpha(g)$ for each $g \in G$; a subset $\rho(a,n)$ for each $a \in \mathcal{W}$ and $n \in N$; and that $\mu(s_i, a)$ indicates whether s_i is true or false at each $a \in \mathcal{W}$.

Given $p \in P$ and $a \in \mathcal{W}$, we define $[\![p]\!]_a$ to be a subset of $\mathcal{W}$, as follows:

- $[\![g]\!]_a = \alpha(g)$ if $g \in G$,
- $[\![n]\!]_a = \rho(n,a)$ if $n \in N$,
- $[\![p'\mathtt{s}\ q]\!]_a = \bigcup\{[\![q]\!]_b \mid b \in [\![p]\!]_a\}$.

For each p, we obtain a relation on $\mathcal{W}$, namely $\{(a,b) \mid b \in [\![p]\!]_a\}$. The relation for $p'\mathtt{s}\ q$ is simply the composition of the relations for p and q.

Given a statement s and $a \in \mathcal{W}$, we write $a \models s$ to mean that s is true at a. The definition of $a \models s$ is:

- $a \models s_i$ is $\mu(s_i, a)$, for $s_i \in S$,
- $a \models (s \wedge t)$ is true iff both $a \models s$ and $a \models t$ are true,
- $a \models (\neg s)$ is true iff $a \models s$ is false,
- $a \models (p \mapsto q)$ is true iff $[\![p]\!]_a \subseteq [\![q]\!]_a$,
- $a \models (p\ \mathit{says}\ s)$ is true iff $b \models s$ is true for all $b \in [\![p]\!]_a$.

Here we are not using the equality interpretation of binding, as the interpretation of $p \mapsto q$ is obviously not symmetric. The interpretation guarantees that q speaks for p in the sense that if q *says* s then p *says* s.

A statement s is *valid* if and only if $a \models s$ is true for all choices of $\mathcal{W}$, α, ρ, and μ, and all $a \in \mathcal{W}$. The following proposition establishes the soundness of the logic of section 3 with respect to the semantics of this section. The converse to this proposition (completeness) is probably false.

Proposition 4 *If s is provable in the logic, then s is valid.*

Soundness implies consistency:

Corollary 5 *If s is provable in the logic, then $\neg s$ is not.*

As a sanity check, we can now show that some undesirable formulas (discussed in section 3) are not derivable in the logic. We show that a formula $\mathtt{DNS!!} \mapsto KL$ cannot be derived from two hypotheses s and t. The first hypothesis, s, lists some "normal" statement that DNS!! makes, binding local names to principal expressions; one may be able to generalize the corollary to include other "normal" statements. The second hypothesis, t, lists statements that KL makes, possibly with the goal of hijacking DNS!!; these statements are completely arbitrary. The corollary guarantees that, despite these arbitrary statements by KL, one cannot derive that DNS!! is bound to KL.

Corollary 6 *Assume that KL and* DNS!! *are distinct global identifiers. Let s be any formula of the form* DNS!! *says* $(n_1 \mapsto p_1) \wedge \ldots \wedge$ DNS!! *says* $(n_k \mapsto p_k)$*, where $n_1, \ldots, n_k$ are local names and $p_1, \ldots, p_k$ are arbitrary principal expressions. Let t be any formula of the form KL says $t_1 \wedge \ldots \wedge KL$ says t_k, where $t_1, \ldots, t_k$ are arbitrary statements. Then the formula $s \wedge t \supset (\texttt{DNS!!} \mapsto KL)$ is not derivable in the logic.*

5. `Self`

In this section, we consider the construct `Self`, which is our notation for `ref` with no arguments, (`ref`:).

Intuitively, `Self` represents the current principal. Therefore, it seems sensible to add `Self` to the set of principal expressions, and to extend the logic with the following four axioms:

Identity:	$(\texttt{Self}'\texttt{s}\ p) \mapsto p$	$p \mapsto (\texttt{Self}'\texttt{s}\ p)$
	$(p'\texttt{s Self}) \mapsto p$	$p \mapsto (p'\texttt{s Self})$

It is also easy to extend the semantics, validating the new axioms. (It suffices to let $[\![\texttt{Self}]\!]_a = \{a\}$ for every $a \in \mathcal{W}$.)

Given the formal properties of `Self`, it may appear as a fairly dull construct. After all, the new axioms show how it can be eliminated from expressions; and the semantics of `Self` is routine.

Nevertheless, the use of `Self` is sometimes convenient. For example, it enables us to write the formula p *says* $(n \mapsto \texttt{Self})$. In the case in which p is a public key K and n is `Ted`, this statement corresponds to a certificate that binds `Ted` to K. The certificate would not need to mention K explicitly. Instead, it can refer to K as `Self`, that is, the public key used for verifying the signature on the certificate. Given K *says* $(\texttt{Ted} \mapsto \texttt{Self})$, we can derive $(K'\texttt{s Ted}) \mapsto (K'\texttt{s Self})$, hence $(K'\texttt{s Ted}) \mapsto K$. In this respect, K *says* $(\texttt{Ted} \mapsto \texttt{Self})$ has the same effect as K *says* $(\texttt{Ted} \mapsto K)$, but the "wire representation" of K *says* $(\texttt{Ted} \mapsto \texttt{Self})$ might be different.

Continuing this example, let us assume that $\texttt{Edward} \mapsto K$ holds in the current name space. From K *says* $(\texttt{Ted} \mapsto \texttt{Self})$ we obtain `Edward` *says* $(\texttt{Ted} \mapsto \texttt{Self})$, and then $(\texttt{Edward}'\texttt{s Ted}) \mapsto \texttt{Edward}$. This is a nice minor result that we would not have reached from K *says* $(\texttt{Ted} \mapsto K)$.

According to our new axioms, $K'\texttt{s Self}$ and K are bound to each other. Therefore, K could say, truthfully, $K \mapsto \texttt{Self}$. Since K speaks for `Edward`, `Edward` would say $K \mapsto \texttt{Self}$ as well. This is a somewhat perplexing consequence of apparently sensible hypotheses: a true statement by K leads to a suspicious statement by `Edward`. On the other hand, `Edward`'s statement should be harmless, since `Edward` should not have the authority to convince others that it speaks for K when in fact `Edward` is bound to K but not vice versa.

In summary, the construct `Self` (in SDSI, (`ref`:)) may be useful occasionally. However, it has the potential to be surprising; like all relative names, it should be handled with care.

6. Conclusions

SDSI's linked local name spaces appear as a promising innovation. This paper attempts to contribute to their understanding, both through informal explanations and through the development of a logic. The logic generalizes the SDSI name-resolution algorithm. It permits not only the evaluation of compound names, but also reasoning about those names and their bindings. Having a logic is not synonymous with security; however, a logic can complement an operational approach, and provide another perspective from which to examine security issues.

Acknowledgements

Ron Rivest made many useful comments on a draft of this paper.

Appendix: Proofs

Proposition 1 *Assume that a set of bindings, each of the form $(n \mapsto q)$, is given for* `cp` *and for each global identifier g. Let E be the conjunction of the formulas $(n \mapsto q)$, for each binding $(n \mapsto q)$ for* `cp`*, and g says $(n \mapsto q)$, for each binding $(n \mapsto q)$ for g. For every principal expression p and global identifiers g and f, we have:*

- *$E \supset (p \mapsto f)$ is provable with the name-resolution rules if and only if* $\texttt{REF2}(\texttt{cp}, p)$ *yields f.*
- *$E \supset ((g'\texttt{s}\ p) \mapsto f)$ is provable with the name-resolution rules if and only if* $\texttt{REF2}(g, p)$ *yields f.*

Proof The "if" direction of the proof is by induction on the execution of $\texttt{REF2}(o, p)$, for both kinds of o together.

- First, let us suppose that the algorithm terminates immediately returning p when p is a global identifier f. In that case, we obtain:
  - $E \supset (f \mapsto f)$ by rule Base case 1,
  - $E \supset ((g'\texttt{s}\ f) \mapsto f)$ by rule Base case 2.
- Next, if p is a local name m, and the assumptions for o include $(m \mapsto q)$, then the algorithm invokes itself recursively, and $\texttt{REF2}(o, q)$ returns f. By induction hypothesis, $E \supset (q \mapsto f)$ is provable when o is `cp`, and $E \supset ((g'\texttt{s}\ q) \mapsto f)$ is provable when o is g. We obtain:

- $E \supset (m \mapsto f)$ by rule Local name 1,
- $E \supset ((g' \mathsf{s}\ m) \mapsto f)$ by rule Local name 2.

- Finally, suppose that p is $(q' \mathsf{s}\ r)$. Then the algorithm invokes itself recursively twice. In the first recursive invocation, $\mathtt{REF2}(o, q)$ yields a result f_0. By induction hypothesis, $E \supset (q \mapsto f_0)$ is provable if o is $\mathtt{cp}$, and $E \supset ((g' \mathsf{s}\ q) \mapsto f_0)$ is provable if o is g. In the second recursive invocation, $\mathtt{REF2}(f_0, r)$ yields the final result f. By induction hypothesis, $E \supset ((f_0' \mathsf{s}\ r) \mapsto f)$ is provable. We obtain:
  - $E \supset ((q' \mathsf{s}\ r) \mapsto f)$ by rule Ref 1,
  - $E \supset ((g' \mathsf{s}\ (q' \mathsf{s}\ r)) \mapsto f)$ by rule Ref 2.

The "only if" direction of the proof is by induction on the proof of $E \supset (p \mapsto f)$ or $E \supset ((g' \mathsf{s}\ p) \mapsto f)$. There is one case for each of the name-resolution rules:

- Base case 1: trivial, since $\mathtt{REF2}(\mathtt{cp}, f)$ yields f.
- Base case 2: trivial, since $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ f))$ and $\mathtt{REF2}(g, f)$ yield f.
- Local name 1: By induction hypothesis, $\mathtt{REF2}(\mathtt{cp}, q)$ yields f. This implies that $\mathtt{REF2}(\mathtt{cp}, n)$ yields f if n is bound to q.
- Local name 2: By induction hypothesis, $\mathtt{REF2}(g, q)$ yields f. This implies that both $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ n))$ and $\mathtt{REF2}(g, n)$ yield f if n is bound to q in g's name space.
- Ref 1: By induction hypothesis, $\mathtt{REF2}(\mathtt{cp}, q)$ yields f_0 and $\mathtt{REF2}(f_0, r)$ yields f, hence $\mathtt{REF2}(\mathtt{cp}, (q' \mathsf{s}\ r))$ yields f. In the case where q happens to be a global identifier h, we also need to prove that $\mathtt{REF2}(h, r)$ yields f. But if q is h and $\mathtt{REF2}(\mathtt{cp}, q)$ yields f_0, then h is f_0, and we already have that $\mathtt{REF2}(f_0, r)$ yields f.
- Ref 2: By induction hypothesis, $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ q))$ and $\mathtt{REF2}(g, q)$ yield f_0, and $\mathtt{REF2}(f_0, r)$ yields f. First we prove that $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ (q' \mathsf{s}\ r)))$ yields f. An invocation of $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ (q' \mathsf{s}\ r)))$ turns into an invocation of $\mathtt{REF2}(\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ q)), r)$; since $\mathtt{REF2}(\mathtt{cp}, (g' \mathsf{s}\ q))$ yields f_0, this turns into an invocation of $\mathtt{REF2}(f_0, r)$, which yields f. Next we prove that $\mathtt{REF2}(g, q' \mathsf{s}\ r)$ yields f. This is an immediate consequence of the facts that $\mathtt{REF2}(g, q)$ yields f_0 and $\mathtt{REF2}(f_0, r)$ yields f.

□

Proposition 2 *The name-resolution rules are sound with respect to the logic. That is, if $E \supset (p \mapsto f)$ is provable using those rules, then it is also provable in the logic.*

Proof The proof is by induction on the proof of $E \supset (p \mapsto f)$ with the name-resolution rules. There is one case for each of the rules.

- Base case 1: by axiom Reflexivity.
- Base case 2: by axiom Globality. (Note that only a special case of Globality is needed: $(f' \mathsf{s}\ g) \mapsto g$.)
- Local name 1: by axiom Transitivity.
- Local name 2: by axioms Linking and Transitivity.
- Ref 1: by axioms Left-monotonicity and Transitivity.
- Ref 2: by axioms Associativity, Left-monotonicity, and Transitivity. (Note that Associativity is used only in one direction, and in a special case: $(f' \mathsf{s}\ (q' \mathsf{s}\ r)) \mapsto ((f' \mathsf{s}\ q)' \mathsf{s}\ r)$.)

□

Proposition 4 *If s is provable, then s is valid.*

Proof The propositional axioms and rules are validated, since the propositional connectives are interpreted in the usual manner. So are the standard axiom and rule of modal logic. Reflexivity and Transitivity are validated, straightforwardly. Associativity holds essentially because the composition of binary relations is associative. The remaining axioms require small arguments:

- For Globality: If g is a global identifier, then $[\![g]\!]_a$ is independent of a (and is determined by α alone); therefore, $[\![p' \mathsf{s}\ g]\!]_a = \bigcup\{[\![g]\!]_b \mid b \in [\![p]\!]_a\} \subseteq [\![g]\!]_a$. Equality holds whenever $[\![p]\!]_a \neq \emptyset$; otherwise, $[\![p' \mathsf{s}\ g]\!]_a = \emptyset$.
- For Linking: If $a \models (p\ \mathit{says}\ (n \mapsto r))$ is true, then $b \models n \mapsto r$ is true for all $b \in [\![p]\!]_a$, and $[\![n]\!]_b \subseteq [\![r]\!]_b$ for all such b. Hence, $\bigcup\{[\![n]\!]_b \mid b \in [\![p]\!]_a\} \subseteq \bigcup\{[\![q]\!]_b \mid b \in [\![p]\!]_a\}$, that is, $[\![p' \mathsf{s}\ n]\!]_a \subseteq [\![p' \mathsf{s}\ r]\!]_a$. Therefore, $a \models (p' \mathsf{s}\ n) \mapsto (p' \mathsf{s}\ r)$ is true.
- For Speaking-for: If $a \models q \mapsto r$ is true, then $[\![q]\!]_a \subseteq [\![r]\!]_a$. Therefore, if $a \models r\ \mathit{says}\ s$ is true, then $b \models s$ is true for all $b \in [\![r]\!]_a$, and $b \models s$ is true for all $b \in [\![q]\!]_a$, so $a \models q\ \mathit{says}\ s$ is true.

□

Corollary 6 *Assume that KL and $\mathtt{DNS!!}$ are distinct global identifiers. Let s be any formula of the form $\mathtt{DNS!!}\ \mathit{says}\ (n_1 \mapsto p_1) \wedge \ldots \wedge \mathtt{DNS!!}\ \mathit{says}\ (n_k \mapsto p_k)$, where $n_1, \ldots, n_k$ are local names and $p_1, \ldots, p_k$ are arbitrary principal expressions. Let t be any formula of the form $KL\ \mathit{says}\ t_1 \wedge \ldots \wedge KL\ \mathit{says}\ t_k$, where $t_1, \ldots, t_k$ are arbitrary statements. Then the formula $s \wedge t \supset (\mathtt{DNS!!} \mapsto KL)$ is not derivable in the logic.*

Proof If the formula in question was derivable, then it would be valid, according to Proposition 4. So we show that it is not valid. It suffices to give a set $\mathcal{W}$, an element $a \in \mathcal{W}$, and functions α, ρ, and μ such that $a \models s \wedge t$ is true while $a \models \texttt{DNS!!} \mapsto KL$ is false. Trivial choices suffice. We let $\mathcal{W}$ be the singleton $\{1\}$, let $a = 1$, let $\alpha(KL) = \emptyset$ and $\alpha(\texttt{DNS!!}) = \mathcal{W}$, and let $\rho(n_i, a) = \emptyset$ for all local names n_i. (Any remaining properties of α, ρ, and μ are irrelevant.) We have that $a \models s$ is true, since $\rho(n_i, a) = \emptyset$, that $a \models t$ is true, since $\alpha(KL) = \emptyset$, and that $a \models \texttt{DNS!!} \mapsto KL$ is false, since $\alpha(\texttt{DNS!!}) \not\subseteq \alpha(KL)$. □

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, Oct. 1993.

[2] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 164–173, May 1996.

[3] CCITT. *Blue Book (Recommendation X.509 and ISO 9594-8: The directory-authentication framework)*. CCITT, 1988.

[4] C. M. Ellison, B. Frantz, and B. M. Thomas. Simple public key certificate. Internet draft, at `http://www.clark.net/pub/cme/spki.txt`, 1996.

[5] M. Gasser, A. Goldstein, C. Kaufman, and B. Lampson. The Digital Distributed System Security Architecture. In *Proceedings of the 1989 National Computer Security Conference*, pages 305–319, Oct. 1989.

[6] R. Goldblatt. *Logics of Time and Computation*. Number 7 in CSLI Lecture Notes. CSLI, Stanford, 1987.

[7] A. J. Grove and J. Y. Halpern. Naming and identity in epistemic logics, I: The propositional case. *Journal of Logic and Computation*, 3(4):345–378, 1993.

[8] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, Nov. 1992.

[9] R. L. Rivest and B. Lampson. SDSI — A Simple Distributed Security Infrastructure. Version 1.1, at `http://theory.lcs.mit.edu/~rivest/sdsi11.html`, October 2, 1996.

[10] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, Feb. 1994.

[11] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.