Authentication Primitives for Refining Protocol Specifications *

Chiara Bodei, Pierpaolo Degano
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, I-56125 Pisa, Italy
{chiara,degano}@di.unipi.it

Riccardo Focardi
Dipartimento di Informatica, Università Ca' Foscari di Venezia
Via Torino 155, I-30173 Venezia, Italy
focardi@dsi.unive.it

Corrado Priami
Dipartimento di Informatica e Telecomunicazioni,
Università di Trento
Via Sommarive, 1438050 Povo (TN), Italy
priami@science.unitn.it

Abstract

We propose a way to abstract from various specifications of authentication and to obtain idealized protocols "secure by construction". This feature enables us to prove that a cryptographic protocol is the correct implementation of the corresponding abstract protocol. Our proposal relies on the combination of two authentication primitives, proposed by the authors in [20, 18] to a simplified version of the spi calculus.

Introduction Authentication is one of the main issues in security and it can have different purposes depending on the specific application considered. For example, *entity authentication* is related to the verification of an entity's claimed identity [1], while *message authentication* should make it possible for the receiver of a message to ascertain its origin [2]. In recent years there have been some formalizations of these different aspects of authentication (see, e.g., [3, 4, 5, 6, 7, 8, 9, 16]). These formalizations are crucial for proofs of authentication properties, that sometimes have been automatized (see, e.g. [10, 11, 12, 13, 14]).

*This work has been partially supported by MURST Progetto TOSCA, Progetto AI, TS & CFA and Progetto "Metodi formali per la Sicurezza".

We here slightly extend the spi calculus [3, 15], a concurrent language with cryptographic primitives, based on the π-calculus [17]. We give this calculus certain kinds of semantics, exploiting the built-in mechanisms for authentication, introduced in [18] Our mechanisms enable us to abstract from the various implementations/specifications of authentication, and to obtain idealized protocols which are "secure by construction". Our protocols, or rather their specifications can then be seen as a reference for proving the correctness of "real protocols".

In particular, our first mechanism, called *partner authentication* [18], guarantees each principal A to engage an entire run session with the same partner B. Essentially, the semantics provides a way of "localizing" a channel, say c, to A and B, so that the partners accept sensitive communications on the localized c, only.

Our localization relies on the so-called *relative address* of A with respect to B [19]. Intuitively, this represents the path between A and B in (an abstract view of) the network (as defined by the syntax of the calculus). Relative addresses are *not* available to the users of the calculus: they are used by the abstract machine of the calculus only, defined by its semantics. Therefore, relative addresses cannot be forged by intruders.

Also our second mechanism, called *message authentication* [20, 18], exploits relative addresses: a datum belonging to a principal A is seen by B as "localized' in the memory space of A own. So, our primitive enables the receiver of a message to ascertain its origin, i.e. the process that created it.

A typical approach for proving authentication properties presented in the literature is the following. First, a protocol is specified in a certain formal model. The specification is then shown to enjoy the desired properties, regardless of its operating environment, that can be unreliable, and can even harbour a hostile intruder. We propose to proceed as follows. An *abstract* version of a protocol is written using the above sketched primitives, and therefore it has the desired authentication properties "by construction". Then, we compare the behaviour of a different, more *concrete* version of the protocol – possibly involving encryptions, nonces, signatures and the like — with the behaviour of the abstract protocol. This is done using testing equivalence, a well-known technique on process algebras.

Our notion directly derives from the Non-Interference notion called NDC that has been applied to protocol analysis in [7, 6]. Note also that the idea of comparing cryptographic protocol with secure-by-construction specifications is also similar to the one proposed in [3] where a protocol is compared with "its own" secure specification. We are indeed refining Abadi's and Gordon's approach [3]: the secure abstract protocol here is unique (as we will show in the following) and based on abstract authentication primitives. On the contrary, in [3] for each protocol one needs to derive a secure specification (still based on cryptography) and to use it as a reference for proving authentication.

In the following, we intuitively survey the spi calculus, a basic calculus for modelling concurrent and mobile agents. Then, we shall introduce the extensions necessary for dealing with our authentication primitives.

The Spi Calculus In this section we intuitively recall a simplified[1] version of the spi calculus [3, 15]. This calculus extends the π-calculus [17], with cryptographic primitives. Here, terms can be names or variables and can also be structured as encryptions $\{M_1, \ldots, M_k\}_N$. An encryption $\{M_1, \ldots, M_k\}_N$ represents the ciphertext obtained by encrypting $M_1, \ldots, M_k$ under the key N, using a shared-key cryptosystem such as DES [21]. Most of the processes constructs should be familiar from earlier calculi: I/O constructs, parallel composition, restriction, matching, replication. We give below the syntax and, afterwards, we intuitively present the dynamics of processes. Terms and processes are defined according to the following BNF-like grammars.

$L, M, N ::=$	**terms**
a, b, c, k, m, n	*names*
x, y, z, w	*variables*
$\{M_1, \ldots, M_k\}_N$	*shared encryption*
$P, Q, R ::=$	**processes**
$\mathbf{0}$	*nil*
$\overline{M}\langle N\rangle.P$	*output*
$M(x).P$	*input*
$(\nu m)P$	*restriction*
$P \vert P$	*parallel composition*
$[M = N]P$	*matching*
$!P$	*replication*
$case\ L\ of\ \{x_1, \ldots, x_k\}_k\ in\ P$	*shared−key decryption*

- The null process $\mathbf{0}$ does nothing.
- The process $\overline{M}\langle N\rangle.P$ sends the term N on M, provided that there is another process waiting to receive on the same channel. Then behaves like P. Here and below M should be a name or a variable: no process can use an encryption as a channel.
- The process $M(x).P$ is ready to receive an input N on channel m and to behave like $P\{N/x\}$, where the term N is bound to the variable x.
- The operator $(\nu m)P$ acts as a static declaration (i.e. a binder for) the name m in the process P that it prefixes. The agent $(\nu m)P$ behaves as P except that actions on m are prohibited.
- The operator $|$ describes parallel composition of processes. The components of $P|Q$ may act independently; also, an output action of P (resp. Q) at any output port $\overline{M}$ may synchronize with an input action of Q (resp. P) at M. In this case, a silent action τ results.
- Matching $[M = N]P$ is an `if-then` operator: process P is activated only if $M = N$.

[1] In the full calculus, terms can also be pairs, zero and successors of terms. Extending our proposal to the full calculus is easy.

- The process $!P$ behaves as infinitely many copies of P running in parallel.
- The process $case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P$ attempts to decrypt L with the key N. If L has the form $\{M_1, \ldots, M_k\}_N$, then the process behaves as $P\{M_1/x_1, \ldots, M_k/x_k\}$. Otherwise the process is stuck.

To give the flavour of the semantics, we illustrate the dynamic evolution of a simple process S.

Example 1

$$\begin{array}{rcl} S & = & !P \mid Q \\ P & = & \overline{a}\langle\{M\}_k\rangle.\mathbf{0} \\ Q & = & a(x).case\ x\ of\ \{y\}_k\ in\ Q' \\ Q' & = & (\nu h)(\overline{b}\langle\{y\}_h\rangle.\mathbf{0} \mid R) \end{array}$$

$!P$ represents a source of outputs on a *of the message M encrypted under k, as many as needed. The first action of Q is reading a message on channel a. So, we have the following transition $S \xrightarrow{\tau} \mathbf{0} \mid !P \mid case\ \{M\}_k\ of\ \{y\}_k\ in\ Q'$, where $!P$ replicated itself in $(\overline{a}\langle\{M\}_k\rangle.\mathbf{0} \mid !P)$ and $\{M\}_k$ replaced x in (the residual of) Q. Now $\{M\}_k$ can be decrypted, as (the residual of) Q attempts to decrypt with the key k. The system then reads as: $\mathbf{0} \mid !P \mid (\nu h)(\overline{b}\langle\{M\}_h\rangle.\mathbf{0} \mid R)$. The message M is then encrypted with the key h, private to Q' and it is possibly forwarded to R.*

Relative Addresses To present our authentication mechanisms [18], it is convenient to briefly recall the central notion of *relative address* of a process P with respect to another process Q within a network of processes, described in our calculus. A relative address represents the path between P and Q in (an abstract view of) the network (as defined by the syntax of the calculus). More precisely, consider the abstract syntax trees of processes, built using the binary parallel composition as the main operator. Given a process R, the nodes of its tree (see e.g. Fig. 1) correspond to the occurrences of the parallel operator in R, and its leaves are the sequential components of R (roughly, those processes whose top-level operator is a prefix or a summation or a replication). Assuming that the left (resp. right) branches of a tree of sequential processes denote the left (resp. right) component of parallel compositions, then label their arcs with tag $||_0$ (resp. $||_1$).

For instance, in Fig. 1, the address of P_3 relative to P_1 is $l = ||_0||_1 \bullet ||_1||_1||_0$ (read the path upwards from P_1 to the minimal common predecessor and reverse, then downwards to P_3). So to speak, the relative address points back from P_1 to P_3. Note that the relative address of P_3 with respect to P_1 is $||_1||_1||_0 \bullet ||_0||_1$ that we write also as l^{-1}. Relative addresses are then strings written $\vartheta \bullet \vartheta'$, made of $||_0$'s and $||_1$'s.

We are now ready to introduce our primitives that induce a few modifications to the calculus surveyed above. Note that we separately present below the two primitives, but they can be easily combined, in order to enforce both kinds of authentication.

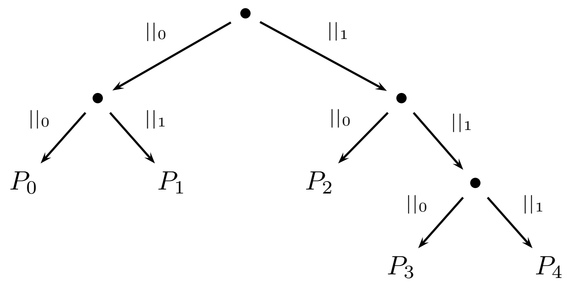

Figure 1: The tree of (sequential) processes of $(P_0|P_1)|(P_2|(P_3|P_4))$.

Partner authentication We can now intuitively present our first semantic mechanism reminiscent of [18]. Essentially, we bind sensitive inputs and outputs to a relative address, i.e. a process P can accept communications on a certain channel, say c, only if the relative address of its partner is equal to an a-priori fixed address l. More precisely, channels may have a relative address as index, and assume the form c_l. Now, our semantics will ensure that P communicates with Q on c_l if and only if the relative address of P with respect to Q is indeed l (and that of Q with respect to P is l^{-1}). Notably, even if another process $R \neq Q$ possesses the channel c_l, R cannot use it to communicate with P, because it will not be placed at the same "location" as Q. Consequently, the hostile process R can never interfere with P and Q while they communicate, as the relative address of R with respect to Q (and to P) is not l (or l^{-1}).

Not always processes know which are the partners' relative addresses. So, we shall also index a channel with a variable λ, to be instantiated by a relative address, only. Whenever a process P, playing for instance the role of sender or initiator, has to communicate for the first time with another process S in the role, e.g. of server, it uses a channel c_λ. Our semantic rules will take care of instantiating λ with the address of P relative to S during the communication. From that point on, P and S will keep communicating for the entire session, using their relative addresses.

Suppose, for instance, that in Fig. 1 the process P_3 is $\overline{a_l}\langle b\rangle.P_3'$ and P_1 is $a_\lambda(x).P_1'$, and recall that the relative address of P_3 with respect to P_1 is $l = ||_1||_1||_0\bullet||_0||_1$. Here P_3 knows the partner address, while P_1 does not. More precisely, for P_3 the output can only match an input executed by the process reachable from P_3 through the relative address l, while the variable λ will be instantiated, during the communication, to the address l^{-1} of the sender P_3, with respect to the receiver P_1. From this point on and for the rest of the protocol, P_1 can use the channel $a_{||_0||_1\bullet||_1||_1||_0}$ (and those with λ instantiated in the same way) to communicate with P_3, only.

Message Authentication Our second mechanism, called *message authentication* [20, 18], enables the receiver of a message to ascertain its origin, i.e. the process that created it. Again it is based on relative addresses.

We illustrate this further extension originally modelled in [20] through a simple example. Suppose that P_3 in Fig. 1 is now $(\nu n)\overline{a}\langle n\rangle.P'_3$. It sends its private name n to $P_1 = a(x).P'_1$. The process P_1 receives it as $||_1||_0\bullet||_1||_1||_0 n = l^{-1}n$. In fact, the name n is enriched with the relative address of P_3, its sender and creator, with respect to its receiver P_1 and the address l^{-1} acts as a reference to P_3. Now suppose that P'_1 forwards to P_2 the name just received, i.e. $l^{-1}n$. We wish to maintain the identity of names, i.e., in this case, the reference to P_3. So, the address l^{-1} will be substituted by a new relative address, that of P_3 with respect to P_2, i.e. $||_1||_0\bullet||_0$. Thus, the name n of P_3 is correctly referred to as $||_1||_0\bullet||_0 n$ in P_2. This updating of relative addresses is done through a suitable address composition operation.

We can now briefly recall our second authentication primitive, $[lM \overset{@}{=} l'N]$, akin to the matching operator. This "address matching" is passed only if the relative addresses of the two localized terms under check coincide, i.e. $l = l'$. For instance, if $P_3 = (\nu d)\overline{a}\langle d\rangle.P'_3$, $P_0 = (\nu b)\overline{a}\langle b\rangle$ and $P_1 = a(x).[x = ||_0||_1\bullet||_1||_1||_0 d]P'_1$, then P'_1 will be executed only if x will be replaced with a name coming from P_3, such as $||_0||_1\bullet||_1||_1||_0 n$. In fact, if P_1 communicates with P_0, then it will receive b, with the address $||_0||_0\bullet||_1||_1||_0$ and the matching cannot be passed.

Implementing Authentication Below, we refer to a notion of protocol implementation, technically based on testing equivalence, which provides a formal way of proving that a certain protocol P implements an abstract protocol P' regardless of the particular message exchange. An *abstract* version of a protocol is written using the above sketched primitives, and therefore it has the desired authentication properties "by construction". Then, we check the abstract protocol against a different, more *concrete* version, possibly involving standard cryptographic operations (e.g. encryptions, nonces).

As authentication primitives provide secure-by-construction (abstract) protocols, the idea is to try to implement them by using, e.g., cryptography. In doing that we have to face the problem of comparing protocols which may heavily differ in the messages exchanged. Indeed, the comparison focuses on the effects of the protocol execution on the continuation, i.e., on what happens after the protocol has been executed. Moreover, since authentication violations are easily revealed by observing the address of the received message, we can exploit our operator of address matching to this aim. In particular, we define a notion of testing equivalence where testers have the ability of directly comparing message addresses (through address matching), thus detecting the origin of messages.

As a matter of fact, here we push a bit further Abadi and Gordon's [3] idea of considering correct a protocol if the environment cannot have any influence on its continuation. More precisely, let $P_s = A_s|B_s$ be an abstract secure-by-construction protocol and $P = A|B$ be a (bit more) concrete (cryptographic) protocol. Suppose also that both B and B_s, after the execution of the protocol, continue with some activity, say B'. Then we require that an external observer should not detect any difference on the behaviour of B' if an intruder E attacks the protocols. In other words, for all intruders E, we require that $A|B|E$ is equivalent to $A_s|B_s|E$. When this holds we say that P *securely implements* P_s. In doing this, we propose to clearly separate the observer, or tester T, from the intruder E. In particular, we let the tester T interact only with the continuation B'. Conversely, we assume that the intruder attacks the protocol, only, and has no interest for what happens later on. This allows to completely abstract from the specific message exchange (i.e., from the communication) and focus only on the "effects" of the protocol execution. In this way we can compare protocols based on cryptography (e.g., P) with secure-by-construction specifications based on abstract primitives (e.g., P_s).

Our notion is such that if P' is a correct-by-construction protocol, specified through our authentication primitives, and P securely implements P', then also the behaviour of P in every hostile environment will be correct.

An Example We show how our approach can be applied on one example, admittedly very simple. However, we feel that the ideas and techniques presented could easily scale up also to more complicate protocols.

Consider a simple protocol where A sends a freshly generated message M to B and suppose that B requires authentication of the message, i.e., that M is indeed sent by A. We abstractly denote this as follows:

$$\text{Message 1} \quad A \quad \stackrel{auth}{\rightarrow} \quad B \quad : \; M$$

Note that, if B wants to be guaranteed that he is communicating with A, he needs some trusted information regarding A to be used as a reference. In real protocols this is achieved, e.g., through a password or key known by A only. We use instead the location of the entity that we want to authenticate. We are interested in a multi-session version of the protocol, in which m copies of the sender A want to communicate with m copies of the receiver B.

To specify this abstract protocol by exploiting our partner authentication primitive, we define a startup primitive that, exchanges the processes locations in a trusted way. Such a primitive is indeed just a macro, defined as follows:

$$m\_startup(t_P, P, t_Q, Q) \quad = \quad (\nu s)(\ !\overline{s}_{t_P}\langle * \rangle.P \mid !s_{t_Q}(x).Q\ )$$

where the two processes that initiate the startup by a communication over s are replicated through the "!"operator. Note that here we have many instances of the two location variables λ_P and λ_Q. We can prove that, given a process $m\_startup(\lambda_P, P, \lambda_Q, Q) \mid E$, in any possible execution the two location variables λ_P and λ_Q can only be assigned to the relative address of one instance of Q with respect to one instance of P and of one instance of P with respect to one instance of Q, respectively. Moreover, two location variables, arising from two different sessions, never point to the same process. We now define the multi-session specification as follows:

$$
\begin{array}{rcl}
P^m & = & m\_startup(\epsilon\bullet\epsilon, A, \lambda_A, B) \\
A & = & (\nu M)\overline{c}\langle M\rangle \\
B & = & c_{\lambda_A}(z).B'(z)
\end{array}
$$

After the startup phase, each copy of B waits a message z exactly from the location of one copy of A, thus this protocol is secure-by-construction. Note that A is not locating the output of the message, as our aim here is to model authentication. Locating the output as well would give a secrecy guarantee on the communication of M. As a matter of fact, a located output would assure that the receiver is exactly the intended one, i.e., no malicious intruder could intercept any message M sent on the located channel.

It is possible to prove that P^m enjoys the following two properties.

Authentication Each copy of B receives a message enriched with the relative address of **one** of the copies of A.

Freshness For every pair of copies of B, the two messages received have been originated by **two different** copies of the process A.

Consider now a specification that uses cryptography to provide authentication of the message, like in:

$$
\text{Message 1} \quad A \quad \rightarrow \quad B \quad : \{M\}_{K_{AB}}
$$

where K_{AB} is an encryption key shared between A and B. We specify this protocol as follows:

$$
\begin{array}{rcl}
P_2^m & = & (\nu K_{AB})(!A_2 \mid !B_2) \\
A_2 & = & (\nu M)\overline{c}\langle\{M\}_{K_{AB}}\rangle \\
B_2 & = & c(z).case\ z\ of\ \{w\}_{K_{AB}}\ in\ B'(w)
\end{array}
$$

Here, each copy of A_2 encrypts M in order to guarantee that no one else is able to substitute it with a different message. Nevertheless, we can prove that P_2^m does not implement P^m. The process P_2^m is indeed prone to replay attacks, as it can easily be observed. More in detail, consider an intruder that intercepts one encrypted message and replays it later: $E = c(z).\overline{c}\langle z\rangle.\overline{c}\langle z\rangle$. It is easy to see that $P_2^m \mid E$ may evolve to $(\nu K_{AB})(\mathbf{0} \mid !A_2 \mid B'(M) \mid B'(M) \mid !B_2)$, where two copies of B_2 has accepted the same message. As M is the same for both the copies, we have that the *Freshness* property above does not hold (which is instead satisfied by P^m) and this process does not securely implements P^m.

We end this section by giving a correct implementation of the multi-session authentication protocol P^m, which exploits a typical challenge-response mechanism to guarantee authentication:

$$
\begin{array}{llcll}
\text{Message 1} & B & \rightarrow & A & : N \\
\text{Message 2} & A & \rightarrow & B & : \{A, M, N\}_{K_{AB}}
\end{array}
$$

where N is a freshly generated nonce that constitutes the challenge. It can be formally specified as follows:

$$\begin{array}{rcl} P_3 & = & (\nu K_{AB})(A_3 \mid B_3) \\ A_3 & = & (\nu M)c(ns).\overline{c}\langle\{A, M, ns\}_{K_{AB}}\rangle \\ B_3 & = & (\nu N)\overline{c}\langle N\rangle.c(x).case\ x\ of\ \ \{id, z, w\}_{K_{AB}}\ in\ [id = A][w = N]B'(z) \end{array}$$

Note that now no replay attack is possible as the fresh nonce is always checked by B against the second parameter of the encrypted message received. [2] This repairs the problem of protocol P_2^m and should make P_3 a secure implementation of P^m.

References

[1] International Organization for Standardization. Information technology - Security techniques - Entity authentication mechanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.

[2] B. Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 1996. Second edition.

[3] M. Abadi and A. D. Gordon. "A Calculus for Cryptographic Protocols: The Spi Calculus". *Information and Computation*, 148(1):1–70, January 1999.

[4] M. Burrows, M. Abadi, and R. Needham. "A Logic of Authentication". *ACM Transactions on Computer Systems*, pp. 18–36, February 1990.

[5] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. "Strand Spaces: Why is a Security Protocol Correct?". In *Proc. of the 1998 Symposium on Security and Privacy.* IEEE Computer Society Press, May 1998.

[6] R. Focardi, R. Gorrieri, and F. Martinelli. "Non Interference for the Analysis of Cryptographic Protocols". In *Proc. of ICALP'00*, LNCS 1853, Springer, 2000.

[7] R. Focardi and F. Martinelli. "A Uniform Approach for the Definition of Security Properties". In *Proc. of World Congress on Formal Methods in the Development of Computing Systems*, LNCS 1708, pp. 794–813, Springer-Verlag, 1999.

[8] G. Lowe. "A Hierarchy of Authentication Specification". In *Proc. of the 10th Computer Security Foundation Workshop.* IEEE press, 1997.

[9] S. Schneider. "Verifying authentication protocols in CSP". *IEEE Transactions on Software Engineering*, 24(9), Sept. 1998.

[10] A. Durante, R. Focardi, and R. Gorrieri. "A Compiler for Analysing Cryptographic Protocols Using Non-Interference". To appear on ACM TOSEM.

[11] R. Focardi, A. Ghelli, and R. Gorrieri. "Using Non Interference for the Analysis of Security Protocols ". In *Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, DIMACS Center, Rutgers University, 1997.

[12] R. Kemmerer, C. Meadows, and J. Millen. "Three systems for cryptographic protocol analysis". *J. Cryptology*, 7(2):79–130, 1994.

[13] G. Lowe. "Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR". In *Proc. of TACAS'96*, LNCS 1055, pp. 146–166, Springer-Verlag, 1996.

[2] The identifier A in the second message has the aim of breaking the symmetry thus avoiding dangerous *reflection* attacks, i.e., attacks in which the intruder exploits the fact that B is playing both the initiator and the responder roles to obtain the nonce encrypted with the correct key K_{AB}. As in this protocol specification B is only playing the responder role, this modification is not strictly necessary.

[14] J. C. Mitchell, M. Mitchell, and U. Stern. "Automated Analysis of Cryptographic Protocols Using Murϕ". In *Proc. of the 1997 IEEE Symposium on Research in Security and Privacy*, pp. 141–153. IEEE Computer Society Press, 1997.

[15] M. Abadi. 'Secrecy by Typing In Security protocols". *Journal of the ACM*, 5(46):18–36, sept 1999.

[16] M. Abadi, C. Fournet, G. Gonthier. Authentication Primitives and their compilation. In *Proc. of POPL'00*, pp. 302–315. ACM Press, 2000.

[17] R. Milner, J. Parrow, and D. Walker. "A Calculus of Mobile Processes (I and II)". *Information and Computation*, 100(1):1–77, 1992.

[18] C. Bodei, P. Degano, R. Focardi, and C. Priami. "Primitives for Authentication in Process Algebras". To appear in *Theoretical Computer Science*. Available at http://www.di.unipi.it/ chiara/publ-40/TCS01.ps.

[19] C. Bodei, P. Degano, and C. Priami. "Names of the π-Calculus Agents Handled Locally". *Theoretical Computer Science*, 253(2):155–184, 2001.

[20] C. Bodei, P. Degano, R. Focardi, and C. Priami. "Authentication via Localized Names". In *Proc. of the 12th Computer Security Foundation Workshop*, pp. 98–110. IEEE press, 1999.

[21] National Bureau of Standards. "Data Encryption Standard (DES)". FIPS Publication 46, 1977.