# Nominal Calculi for Security and Mobility

Andrew D. Gordon
University of Cambridge
Computer Laboratory
http://www.cl.cam.ac.uk/users/adg

February 17, 1997

Needham defines a pure name to be "nothing but a bit pattern that is an identifier, and is only useful for comparing for identity with other bit patterns—which includes looking up in tables in order to find other information" [13]. In this paper, we argue that pure names are relevant to both security and mobility. A *nominal calculus* is a computational formalism that includes a set of pure names and allows the dynamic generation of fresh, unguessable names. We survey recent work on nominal calculi with primitives representing location failure, process migration and cryptography, and suggest areas for further work.

Needham's definition treats pure names as atoms; on the other hand, an impure name has additional structure of some kind. Perhaps it is a combination of component names; or perhaps there are operations to generate one name from another. To illustrate this distinction, consider the treatment of memory pointers in different programming languages. In Java, for instance, memory pointers are pure names. The type system enforces Needham's abstract view of a bit pattern as a pure name; it allows programs to compare pointers for identity or to de-reference a pointer, but denies other operations, such as pointer arithmetic. On the other hand, in C, for instance, a memory pointer is an impure name; we have direct access to the bit pattern representing the pointer.

## The Pi Calculus: A Nominal Process Calculus

To see the relevance of names and nominal calculi to secure mobile code, we begin by considering the pi calculus, a language much smaller than Java or C, and hence a simpler starting point for experimenting with primitives for security or mobility.

The pi calculus of Milner, Parrow and Walker [12] is a nominal calculus in which computations consist of collections of parallel processes equipped with synchronous communication channels. If $P$ and $Q$ are processes, process $P$ *par* $Q$ consists of $P$ and $Q$ running in parallel, possibly communicating with each other or with the outside world along communication channels. Channels are represented by names. In fact, names are the only data that may be communicated on channels.

Although based on a handful of primitives, the pi calculus is astonishingly expressive. Functional programming may be reduced to the pi calculus, since the lambda calculus can be encoded. Similarly, a variety of formulations of imperative, object-oriented and concurrent programming may be reduced to the pi calculus [10, 17, 20]. All these encodings depend on the presence of pure names and name generation.

A *restriction* in the pi calculus is a process of the form *(new n)P*, which means "make a fresh, unguessable name *n*, and then run *P*." Restriction represents name generation in the pi calculus. The name *n* in the restriction is a *bound identifier*, whose scope is *P*. Restricted names are unguessable; you can only learn a restricted name if someone tells you.

A major innovation of the pi calculus is its collection of equational laws for restricted names. (These were inspired, in part, by the earlier work of Engberg and Nielsen [6].) For example, one law asserts that the processes *(new n)(P par Q)* and *P par (new n)Q* are the same, provided that the name *n* does not occur in process *P*. This law allows the scope of a restricted name to vary. Left to right, it allows scope contraction; right to left, it allows scope expansion. A series of applications of this law, together with other laws, allows the scope of a name to expand and contract as it is passed from one process to another during the course of a computation. In this sense, the scope of a restricted name is mobile. The pi calculus was published as a "A Calculus of Mobile Processes" [12] because, unlike almost all previous process calculi, its mobile restrictions can model the passage of knowledge of a communications channel from one process to another.

## Nominal Calculi for Mobility

Several authors observe that stronger senses of process mobility are important for modelling distributed and mobile systems. In summary, the observation is that the primitives of the pi calculus are *location transparent*, that is, its formal semantics is such that if it were correctly implemented in a distributed system, there would be no way to tell where processes or channels reside.

Amadio and Prasad [3] point out that the pi calculus does not directly model location failure, where a location is a node (for instance, a server or a workstation) in a distributed system. Amadio and Prasad propose a new nominal calculus, the pi-L calculus, in which processes and channels have locations, and in which processes may fail. Names identify locations. They use their calculus to verify a distributed algorithm, intended to withstand location failure.

Synchronisation on global channels is the basic communication primitive of the pi calculus. Many authors note that this is an expensive primitive to implement in a distributed system. One response is to trim the most problematic primitives from the original pi calculus to obtain the choice-free asynchronous pi calculus [9, 15]. Another is to drop channel-based communication in favour of other primitives [7].

Let a *mobile system* be a distributed system in which processes may migrate between locations. Languages for mobile systems include Java, Obliq and Telescript. Process migration is not one of the primitives of the pi calculus. There are, however, nominal calculi, without locations, in which processes may be communicated as values [16, 18], and there are recent nominal calculi with process migration as a primitive [2, 8].

## Nominal Calculi for Security

Having considered mobility, we turn now to the relevance of pure names and nominal calculi to security.

Cryptography is an important way of achieving security in distributed and mobile systems. Generation of fresh random numbers, made from enough bits to be effectively unguessable, is an essential primitive. Such random numbers are pure names; all that matters is their identity. They are used in cryptographic protocols as encryption keys, as nonces—stamps used to prove freshness of a message.

Lampson observes that a cryptographic channel—a channel whose content is encrypted—is identified by the key used for encryption. This observation leads in the pi calculus to modelling keys as names, key generation by restriction, and communication on a cryptographic channel by communication on its key. Unfortunately this encoding is too secure in the sense that it fails to model certain attacks on cryptographic communications: there is no possibility of another process eavesdropping on the channel or of transmitting replayed or spoof messages. Moreover, Abadi observes that communication in the pi calculus enjoys *forward secrecy*; the secrecy of completed communications is not compromised by the subsequent disclosure of a previously secret channel.

The spi calculus [1] is a version of the pi calculus that allows communication on a cryptographic channel to be modelled as communication on a public channel of a term that represents a ciphertext. The name of the communications channel is public; the name representing the secret encryption key is restricted. An attacker intent on disrupting a protocol is modelled as an arbitrary process that runs alongside the protocol, and which can send or receive on the public channels used by the protocol. The encoding is realistic in that it allows for message replay, and forward secrecy fails. If you transmit a ciphertext on a public channel and then publish the key, the attacker can obtain the plaintext.

There are varieties of the spi calculus with primitives for shared-key cryptography, hashing and public-key cryptography. The spi calculus has been used to specify and verify secrecy and authenticity properties of several cryptographic protocols. The spi calculus is lower level than some well known formalisms for security protocols, such as the BAN logic [4], but it has the advantage that specifications are executable, and that its formal foundations are clear and precise.

## Researching Security and Mobility using Nominal Calculi

We have surveyed the work of many researchers connecting names, security and mobility. We claim that to research secure mobile code, as well as building experimental implementations, there is much to be gained by experimenting with nominal calculi that model the semantics of such implementations.

We can take the pi calculus as a paradigm: start with a syntax of processes, express behaviour via a primitive operational semantics, and represent name generation by the restriction operator. As we have seen, the pi calculus is based on invulnerable and centralized communication channels. Processes are mobile by convention only. But it has been augmented, experimentally, with more realistic primitives for distribution, process migration and cryptography. In the future, we expect to see more such experiments.

As we have seen, this line of work is already well underway. Questions for further work include the following. How can we apply the body of work on timed concurrent calculi to model the timestamps found in cryptographic protocols? How can we model access control mechanisms, such as capabilities? Can we provide a formal foundation for logics of belief such as BAN [4] via operational models such as the spi calculus? Can we model cryptographic certification of the integrity of mobile code? Can we adapt model-checking tools for the pi calculus [19] to proving properties of secure mobile code?

Here are some stylistic issues:

- Name generation can be avoided in some formalisms for security; for instance, Hoare's CSP, which is not nominal, has been used successfully to find bugs in cryptographic protocols [11]. On the other hand, a formalism representing a programming language for mobile code must be able to model the resources allocated dynamically by any program; it seems hard to achieve this without modelling name generation of some kind.

- The restriction operator models name generation using a bound name. Restriction is rarely used for modelling name generation in nominal calculi based on sequential computation. Witness the sequential fragment of Java studied by Drossopoulou and Eisenbach [5]; it is nominal but name generation is modelled by non-deterministically picking a fresh, free name. Restriction is probably unnecessary for some purposes, such as verifying a type system, but it seems to be rather useful when calculating equational properties of concurrent programs.

- Several researchers studying security or mobility find it more expedient to do so via primitive operational semantics for new constructs than to try to encode the constructs within the pi calculus itself. For now we expect this trend to continue, but we may hope that after a time of experiment more canonical calculi will emerge.

Finally, we expect research into nominal calculi to have several applications:

- Nominal calculi and operational semantics allow clear and precise description of secure mobile code. Experience shows that formal description often exposes bugs in designs and implementations, even when there is no attempt at formal proof.

- Nominal calculi allow for formal proof of properties of secure mobile code. Computer support for formal proof by theorem proving [14] or model checking [11, 19] is getting steadily better. Still, at present such proof is expensive and requires considerable expertise; it is an area with many challenges.

- Nominal calculi support a variety of type systems that allow properties to be proved of code by automatic type checking. Drossopoulou and Eisenbach's work [5] on a small calculus representing important aspects of the Java type system was mentioned earlier. We expect that nominal calculi will be a productive setting for expressing and verifying a variety of type systems for secure mobile code.

### Acknowledgement

Conversations with Alan Jeffrey, Robin Milner, Roger Needham and Peter Sewell helped to clarify some of my initial thoughts on this subject. Martín Abadi, Luca Cardelli, Alan Jeffrey, Ole Jensen, Fabio Massacci, Benjamin Pierce and Ian Stark suggested improvements to a draft. This work was supported by a University Research Fellowship awarded by the Royal Society.

## References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, Apr. 1997. http://www.cl.cam.ac.uk/users/adg/spi.html.

[2] R. M. Amadio. An asynchronous model of locality, failure, and process mobility. Technical report, Laboratoire d'Informatique de Marseille, Feb. 1997.

[3] R. M. Amadio and S. Prasad. Localities and failures. In *Proceedings of 14th FST and TCS Conference, FST-TCS'94*, volume 880 of *Lecture Notes in Computer Science*, pages 205–216. Springer-Verlag, 1994.

[4] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, Feb. 1989.

[5] S. Drossopoulou and S. Eisenbach. Is the Java type system sound? In *FOOL 4, Paris*, Jan. 1997. http://www.cs.indiana.edu/hyplan/pierce/fool/drossopoulou.ps.gz.

[6] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI PB–208, DAIMI, Aarhus University, May 1986.

[7] C. Fournet and G. Gonthier. A calculus of mobile agents. In *Proceedings CONCUR'96*, Aug. 1996.

[8] C. Fournet and G. Gonthier. The reflexive CHAM and the Join-calculus. In *Proceedings POPL'96*, pages 372–385, Jan. 1996.

[9] K. Honda and M. Tokoro. On asynchronous communication semantics. In *ECOOP'91 Workshop on Object-based Concurrent Computing*, volume 512 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[10] C. Jones. A pi-calculus semantics for an object-based design notation. In *CONCUR'93: Fourth International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1993.

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

[12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–40 and 41–77, 1992.

[13] R. M. Needham. Names. In S. Mullender, editor, *Distributed Systems*, pages 89–101. Addison-Wesley, 1989.

[14] L. C. Paulson. Proving properties of security protocols by induction. Technical Report 409, University of Cambridge Computer Laboratory, Dec. 1996. http://www.cl.cam.ac.uk/users/lcp/papers.

[15] B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In T. Ito and A. Yonezawa, editors, *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994)*, number 907 in Lecture Notes in Computer Science, pages 187–215. Springer-Verlag, Apr. 1995.

[16] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992. Available as Technical Report CST–99–93, Computer Science Department, University of Edinburgh.

[17] D. Sangiorgi. Interpreting typed objects into typed pi-calculus. In *FOOL 3, New Brunswick*, 1996. ftp://ftp.dcs.ed.ac.uk/pub/sad/piOCtyped.ps.gz.

[18] B. Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Informatica*, 1993.

[19] B. Victor and F. Moller. The mobility workbench: A tool for the pi-calculus. In D. Dill, editor, *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[20] D. Walker. Objects in the pi-calculus. *Information and Computation*, 116(2):253–271, 1 Feb. 1995.