SPIKY: A nominal calculus for modelling protocols that use PKIs[*]

David Gray
School of Computing
Dublin City University
Dublin, Ireland
dgray@computing.dcu.ie

Benjamin Aziz
Department of Computer Science
University College Cork
Cork, Ireland
b.aziz@cs.ucc.ie

Geoff Hamilton
School of Computing
Dublin City University
Dublin, Ireland
hamilton@computing.dcu.ie

Abstract

In this paper, we present an extension of the spi calculus that incorporates primitives for the retrieval of (un)certified public and private keys belonging to users of PKI-based systems. The extended notation also formalises the notion of process ownership by a PKI user. We also define the operational semantics of the new notation and give examples of PKI-based security protocols and review some of their authenticity properties.

Keywords: public-key infrastructures, nominal calculi, cryptographic protocols

1 Introduction

To specify a security protocol, it is necessary to give details of the exchanges between the entities participating in the protocol. Typically these exchanges will include both plaintext and ciphertext produced using various keys. Much of the literature uses informal notations in which each entity is identified by a name $(A, B, \ldots)$ and the protocol is defined as a series of numbered steps naming the sending and receiving entities. Each step also specifies the data transferred. For example, the following specification defines a 2-step protocol between entities A and B:

$$\begin{array}{lcl} A \rightarrow B & : & A, N_A \\ B \rightarrow A & : & B, N_B, \{N_A\}_{K_{AB}} \end{array}$$

A sends its identity A and a nonce N_A to B in step 1. B responds by sending its identity B, a nonce N_B and A's nonce N_A encrypted by a shared symmetric key K_{AB} to A in step 2.

There are two major shortcomings with this type of notation:

1. The *internal* behaviours of entities are not specified directly and typically such specifications need accompanying natural language text to explain how entities generate and process data. In addition, how keys are generated, protected and handled is normally explained as accompanying natural language text.

[*]This research was partially funded by Enterprise Ireland Strategic Grant ST/2000/94.

2. These notations are informal and have no underlying theory that can be used to prove properties of such protocols.

As an alternative to these informal notations, *process algebras* such as CSP [8] and nominal calculi [7] based on the π-calculus [10, 11, 13] can be used to give formal specifications (or models) for security protocols. With these specifications, each entity is modelled as a process that describes (at some level of abstraction) the entity's behaviour. Because process algebras have underlying theories, it is possible to verify that security protocols exhibit appropriate security properties; for examples see [1, 12].

While process algebras are normally computationally complete, i.e., they can be used to specify behaviour that is Turing computable [14], they do not allow all the behaviour required of a protocol to be captured. For example, it may not be possible to capture requirements on how keys and secret information should be handled by an entity. In addition, there may be behaviour that is too complex to capture succinctly in an algebra or which is not central to the behaviour of the protocol under investigation.

When dealing with public-key cryptography, the correctness of protocols depends on public keys being correctly associated with their owners. It is common in literature (e.g. [1, 2]) to do this informally by using subscripted names for keys. For example, the key k_A might be designated as the public key belonging to the user A. While these sorts of approaches help a reader understand a specification, they are not amenable formal treatment.

In this paper we explore some simple extensions to the spi-calculus [1, 2] that allow us to specify protocols that use *Public Key Infrastructures* (PKIs). These extensions allow us to formalize the binding of public keys to their owners and allow us to give a more complete formal account of how protocols behave. PKIs such as X.509 [16] are designed to allow *public keys* to be securely bound to their owners. In the case of X.509, this is achieved by naming each entity and using *certificates* to bind names to public keys. Our extensions to the spi-calculus are intended to be independent of any particular PKI technology, so we use an abstract view of the functionality of PKIs in general.

In §(2) we give an overview of the spi-calculus and consider some issues relating to its definition. In §(3) we discuss PKIs and introduce *SPIKY*, an extension to the spi-calculus that can be used to specify protocols that use PKIs. In §(4), we present a couple of examples that use SPIKY and finally, §(5) contains some conclusions and directions for future work.

2 Overview of the spi-calculus

In this section we give an overview of the spi-calculus and draw the reader's attention to points that are important for the remainder of this paper. Full definitions of the spi-calculus are given in [1, 2].

2.1 Syntax

Like the π-calculus, the spi-calculus assumes an infinite set of *names* which are used to represent channels, keys etc. However, unlike the π-calculus, the spi-calculus also assumes an infinite set of *variables* which are used as *formal parameters*[1] to various process constructs. We let a, b, c,

[1]In the π-calculus, names are used as formal parameters, for example, in input processes.

$L, M, N ::=$	
$n, m, a, b, c, k \ldots$	name
(M, N)	pair
$x, y, z \ldots$	variable
M^+	public key component of M
M^-	private key component of M
$\{M\}_N$	symmetric encryption
$\{\![M]\!\}_N$	public-key encryption with public key N
$[\!\{M\}\!]_N$	signature with private key N

Figure 1: Syntax of terms

$P, Q, R ::=$	
$\overline{M}\langle N\rangle.P$	output
$M(x).P$	input (scope of x is P)
$P \mid Q$	composition
$(\nu\ n)P$	restriction (scope of n is P)
$!P$	replication
$[M\ \mathbf{is}\ N]P$	match
$\mathbf{0}$	nil
$\mathbf{let}\ (x, y) = M\ \mathbf{in}\ P$	pair splitting (scope of x, y is P)
$\mathbf{case}\ L\ \mathbf{of}\ \{x\}_N\ \mathbf{in}\ P$	symmetric decryption (scope of x is P)
$\mathbf{case}\ L\ \mathbf{of}\ \{\![x]\!\}_N\ \mathbf{in}\ P$	public-key decryption (scope of x is P)
$\mathbf{case}\ L\ \mathbf{of}\ [\!\{x\}\!]_N\ \mathbf{in}\ P$	signature with recovery validation (scope of x is P)
$[\ [\!\{L, M\}\!]_N]P$	signature with appendix validation
$A(M)$	instantiation of the abstraction A

Figure 2: Syntax of processes

k, m and n range over names, and x, y and z range over variables.

The grammar for *terms* is given in Figure 1 and the grammar for *processes* is given in Figure 2. For simplicity, we have left out terms and processes dealing with integer values. In addition, the spi-calculus presented in [1, 2] does not distinguish between *signature with appendix* and *signature with recovery.* $P[M/x]$ represents the capture-free[2] substitution of all free occurrences of the variable x in the process P by the term M.

If a process is *stuck*, it cannot make any further progress and in particular, it cannot engage in communications. Informally, terms and processes have the following meanings:

- The terms M^+ and M^- denote the public and private keys of the key-pair M.
- An *encryption* term $\{M\}_N$ represents the ciphertext produced by the symmetric encryp-

[2]Since we deal with closed processes containing no free variables, a term M should not contain any variables. It may contain names which could be captured by a restriction, but this can be avoided by suitable α-conversions since processes are considered equal up to renaming of bound names and variables.

tion of M using the key N.

- A *public-key encryption* term $\{[M]\}_N$ represents the ciphertext produced by the public-key encryption of M using the public key N.
- A *signature* term $[\{M\}]_N$ represents the digital signature of M using the private key N.
- For a *symmetric decryption* process **case** L **of** $\{x\}_N$ **in** P, if L is of the form $\{M\}_N$ (i.e., it is ciphertext produced using the key N), then the plaintext M is bound to x in P, i.e., the process behaves as $P[M/x]$; otherwise the process is stuck.

  Note that there is an implicit assumption that it is possible to determine that a piece of ciphertext was produced using a particular key. This is possible if the original plaintext is sufficiently redundant or sufficient redundancy is added to the plaintext before encryption.
- For a *public-key decryption* process **case** L **of** $\{[x]\}_N$ **in** P, if N is the private key K^- and L is of the form $\{[M]\}_{K^+}$ (i.e., it is ciphertext produced using the public key corresponding to the private key N), then the plaintext M is bound to x in P, i.e., the process behaves as $P[M/x]$; otherwise the process is stuck.
- For a *signature with recovery validation* process **case** L **of** $[\{x\}]_N$ **in** P, if N is the public key K^+ and L is of the form $[\{M\}]_{K^-}$ (i.e., it is the digital signature produced using the private key corresponding to the public key N), then M is bound to x in P, i.e., the process behaves as $P[M/x]$; otherwise the process is stuck.
- For a *signature with appendix validation* process $[\ [\{L, M\}]_N]P$, if N is the public key K^+ and L is signature $[\{M\}]_{K^-}$, then this process behaves as P; otherwise the process is stuck.
- An *output process* $\overline{M}\langle N\rangle.P$ is ready to output the term N on the channel M and then behave as the process P.

  An *input process* $M(x).Q$ is ready to perform input on the channel M and then behave as $Q[N/x]$; where N is the term received. In this case, x can be viewed as a formal parameter that is bound to the term N received over the channel M.

  Communication occurs when there is an input and output on a given channel that can *react* together. In this case the term output by the output process is input by the input process and bound to its formal parameter.
- A *composition* $P \mid Q$ behaves as P and Q running in parallel.
- A *restriction* $(\nu\ n)P$ introduces a new name n that may occur in P and has a scope that is *initially* restricted to P.
- A *replication* $!P$ behaves like infinitely many replicas of P running in parallel.
- A *match* $[M$ **is** $N]P$ behaves like P if M and N are the same[3] term; otherwise the process is stuck.
- The *nil* process $\mathbf{0}$ does nothing.
- For a *pair splitting* process **let** $(x_1, x_2) = M$ **in** P, if M is a pair of the form (N_1, N_2) then N_1 is bound to x_1 and N_2 bound to x_2, i.e., the process behaves as $P[N_1/x_1][N_2/x_2]$; otherwise the process is stuck.

[3]What constitutes "the same term" will be discussed below.

By convention, $\overline{M}\langle N\rangle.\mathbf{0}$ is abbreviated to $\overline{M}\langle N\rangle$. We use $\overline{a}\langle M, N\rangle.P$ as an abbreviation for $\overline{a}\langle (M, N)\rangle.P$ and $a(x, y).P$ as an abbreviation for $a(z).\mathbf{let}\ (x, y) = z\ \mathbf{in}\ P$ where z is not a free variable of P.

An *abstraction*[4] $(x)P$ binds the variable x within the process P. We can view an abstraction as a parameterised process that can be *instantiated* by supplying a term for its parameter x. For example, $((x)P)(M)$ represents the *application* of $(x)P$ to the term M and behaves as $P[M/x]$. Abstractions can be used to name parameterised processes by giving definitions of the form $A \triangleq (x)P$ and named processes are instantiated using processes of the form $A(M)$. There are a number of conventions used with abstractions:

- We let $(x_1, \ldots, x_n)P$ stand for $(x_1)\ldots(x_n)P$.
- We let P stand for $()P$.
- We let $A(x) \triangleq P$ stand for $A \triangleq (x)P$.

We write $\mathbf{fn}(M)$ and $\mathbf{fn}(P)$ for the sets of free names in the term M and the process P respectively, and $\mathbf{fv}(M)$ and $\mathbf{fv}(P)$ for the sets of free variables in the term M and the process P respectively. A term or process is *closed* if it has no free variables.

As is typical with nominal calculi, the operational semantics of the spi-calculus is defined by giving a *reaction (reduction) semantics* that captures the internal behaviour of a process independently of its environment, or a *commitment semantics*, which describe interactions with environments. We refer the reader to [1, 2] for details of the reaction and commitment semantics for the spi-calculus.

2.2 Knowledge, Scope and Equality

In nominal calculi, a restriction $(\nu\ n)P$ introduces a new (fresh) name with scope P. Of course, with a concrete representation of a nominal calculus, the same *identifier* may be used in non-overlapping or nested restrictions. For example, the identifier n represents multiple names in the following processes.

$$(\nu\ n)P \mid (\nu\ n)Q \tag{1}$$

$$(\nu\ n)(\overline{c}\langle n\rangle.(\nu\ n)P) \tag{2}$$

$$!(\nu\ n)P \tag{3}$$

Since processes are equal up to the renaming of bound names and variables, we can use α-conversion to avoid name clashes. However, the presence of replication and recursive abstractions makes it necessary to perform renaming dynamically as a process evolves. For example, we can statically rename multiple occurrences of n in (1) and (2), but for (3), since there are an infinite number of occurrences of $(\nu\ n)P$, renaming must occur dynamically as the process evolves.

A unique feature of nominal calculi is *scope extrusion*, i.e., the scope of a name can be extended as a process evolves. In particular, when a name n is output to a process that is not within the current scope of n, the scope of n must be extruded to include this process. In fact, since the semantics of both the π- and spi-calculi include the structural congruence:

[4]There is an obvious analogy with abstractions in the λ-calculus.

$$(\nu\ n)(P \mid Q) \equiv (\nu\ n)P \mid Q \qquad \text{if } n \notin \mathbf{fn}(Q) \tag{4}$$

we can always (possibly after suitable renaming) extrude the scope of any restriction to the outermost level of a composition. However, in this case, after extrusion the process Q will still have no free occurrences of n and will only acquire *knowledge* of n if there is a suitable communication involving n.

Clearly, given the presence of scope extrusion, the scope of a name cannot be used directly as a mechanism for determining the visibility or use of a name within processes, we must also consider whether the process has knowledge of the name. In the case of the π-calculus, a process P has knowledge of a name n if $n \in \mathbf{fn}(P)$ and as a process evolves it may acquire knowledge of a name as the result of an input. Therefore, if we wish to establish that a process P can never know a particular name n, we need to establish the *invariant* $n \notin \mathbf{fn}(P')$ for all processes P' which may evolve from P (in a given context).

When dealing with security, a more challenging problem is establishing that **no** process can acquire knowledge of a particular name. For example, in the process

$$((\nu\ c)((\nu\ n)\overline{c}\langle n\rangle.\mathbf{0}) \mid c(x).\mathbf{0}) \mid R \tag{5}$$

it can be established that there is no process R that can gain knowledge of n. In some cases type checking or simple static analysis [5] can be used to determine that a process cannot have knowledge of a name, but in general, this is not a static property of a process.

For readers familiar with scoping in programming languages, restrictions and scope extrusion in nominal calculi can appear somewhat confusing. In particular, since for simplicity, names in the π-calculus serve as formal parameters (which are effectively programming variables) in input prefixes, some names effectively have static scopes similar to variables in a programming language. For example, consider the π-calculus process:

$$(\nu\ n)c(n).P \tag{6}$$

The name n is restricted to P and this process can only evolve by substituting some other name (input over channel c) for n. Thus, even if the scope of n is extruded, no other process can ever have knowledge of n.

In the spi-calculus, variables are used as formal parameters (see Figure 2) and variables have static scope that cannot be extruded. Thus, in the spi-calculus, we can always view names as *unique values* that are created dynamically as processes evolve. In particular, each name (up to renaming) always represents the same value and is never replaced by another name as processes evolve. In addition, since every name is a unique value, a match $[\alpha\ \mathbf{is}\ \beta]P$ where α and β are names, is stuck unless α and β are the same name.

In the pure π-calculus, only names are communicated between processes, but the spi-calculus extends communication to terms that also consist of pairs and ciphertext. This requires us to extend the concepts of knowledge and equality.

2.2.1 Knowledge

If a name is encrypted as part of some ciphertext, then a process will only have knowledge of that name if it decrypts the ciphertext. For any given process we can determine if decryption occurs, but in general we are more interested in determining properties of arbitrary processes. As before, we may wish to establish that no intruder process can decrypt the ciphertext and gain knowledge of the name.

For example, given the definitions

$$\begin{aligned} A &\triangleq (\nu\ n)\overline{c}\langle\{n\}_k\rangle.Q_1 && (7)\\ B &\triangleq c(x).\textbf{case}\ x\ \textbf{of}\ \{y\}_k\ \textbf{in}\ Q_2 && (8)\\ P &\triangleq (\nu\ c)((\nu\ k)(A \mid B)) \mid R) && (9)\end{aligned}$$

A and B are two processes that share a secret key k and communicate over an open channel c. R is an arbitrary intruder process that can intercept communications on c. This arrangement is modelled by the process P in which the scope of the key k is restricted to A and B, but the scope of the channel c includes A, B and R. Initially, R may have knowledge of c but not of k. However, as we wish k to be a secret key shared by A and B, we must ensure that R can never acquire knowledge of k via a communication as P evolves. When P is executed, A creates a new name n, encrypts it using k and sends the resultant ciphertext over the channel c to B. R can capture the ciphertext $\{n\}_k$, but since it never has knowledge of k, it can never discover n.

It is interesting to note that as P evolves, the scope of both k and n may be extruded to include R even though R can never have knowledge of either k or n. For example, given $R \triangleq c(x).\overline{c}\langle x\rangle$, then R can intercept the communication from A over c and give the reduction:

$$P \longrightarrow (\nu\ c, n, k)(Q_1 \mid B \mid \overline{c}\langle\{n\}_k\rangle) \qquad (10)$$

2.2.2 Equality

Encryption schemes[5] may either be *deterministic* or *randomised* [9]. With a deterministic scheme the same plaintext and key will always produce the same ciphertext, e.g., DES in ECB mode is deterministic. With a randomised scheme the same plaintext and key will produce different ciphertexts each time the scheme is applied, e.g., DES in CBC mode is randomised as we do not consider the Initialisation Vector (IV) to be part of the key.

Given two identical ciphertexts, they will have been produced by two applications of a deterministic scheme using the same plaintext and key, or they will be copies of the ciphertext produced by a single application of a scheme[6]. However, given two different ciphertexts produced by a randomised scheme, they may represent the encryption of the same plaintext with the same key.

In their presentations of the spi-calculus [1, 2], Abadi & Gordon specify that a match $[M\ \textbf{is}\ N]P$ behaves as P if the terms M and N are the ***same***. As we have seen, for terms representing names, M and N must be the same name and for terms representing pairs, we can use element-wise equality. However, for terms representing ciphertexts, Abadi & Gordon do not

[5]We distinguish between a basic cipher (e.g., DES) and a scheme that gives details of how such a cipher can be used in practice. For example, in the case of a symmetric cipher, when encrypting large amounts of data, an encryption scheme will specify the *mode of operation* to be used. Modes include Electronic Codebook (ECB) and Cipher-Block Chaining (CBC).

[6]There is a possibility that a ciphertext is some random data produced by an attacker, but provided the encryption scheme adds sufficient redundancy to the plaintext before encryption, it is computationally infeasible for an attacker to generate random data that can be correctly decrypted. It is also possible, but again computationally infeasible, that a ciphertext was produced using a different key.

give an explicit definition of what constitutes a match (although later works, such as [6], seem to adopt a randomised view of ciphertexts).

Equality of the terms $\{M_1\}_{k_1}$and $\{M_2\}_{k_2}$ can be defined in a number of ways:

1. **Strong equality**: $\{M_1\}_{k_1} = \{M_2\}_{k_2}$ if $M_1 = M_2$ and $k_1 = k_2$.
2. **Ciphertext equality**: $\{M_1\}_{k_1} = \{M_2\}_{k_2}$ if $\{M_1\}_{k_1}$ and $\{M_2\}_{k_2}$ are the same ciphertext.
3. **No equality**: $\{M_1\}_{k_1} = \{M_2\}_{k_2}$ is always considered false.

For deterministic schemes, strong and ciphertext equality are identical, but for randomised schemes they are different since strongly equal terms may yield different ciphertexts.

When dealing with the meaning of a process, strong equality is an appropriate definition of equality and it can be used in the definition of bisimilarity. However, because of randomised encryption schemes, this definition of equality is non-computable and is therefore inappropriate for defining a match.

Ciphertext equality is computable but when used to define a match, it may make the behaviour of a process depend on whether we are using a deterministic or randomised encryption scheme. This makes it difficult to reason about the the behaviour of processes and leads to a situation in which testing equivalence is more fine-gained than bisimilarity.

Given the problems with strong and ciphertext equalities, we select option 3 for the semantics of a match; any attempt to compare two encryption terms becomes stuck. Of course, this does not reflect reality as we cannot capture an intruder's ability to compare ciphertexts. However, since in practice ciphertexts are rarely the same[7], we will ignore this problem.

3 SPIKY

In our work we are interested in modelling protocols that use PKIs. Since by their very nature, PKIs introduce the concept of *the owner of a key-pair*, the spi-calculus is not an appropriate notation for modelling such protocols. In particular, the spi-calculus does not have any concept of a user nor of a process acting on behalf of a user.

In this section we first look at the basic properties of PKIs as a motivation for SPIKY, an extension of the spi-calculus that supports PKIs. We then present the syntax and operational semantics of SPIKY.

3.1 Public Key Infrastructures

The main purpose of a PKI is to bind public keys to their owners. Different PKI technologies achieve this in different ways, but typically PKI technologies (such as X.509[16]) have the following two features:

1. A *naming scheme* that allows users of the PKI to be identified. For example, X.509 uses *X.500 Distinguished Names* [15].

[7]For example, nonces are widely used to make ciphertexts different.

2. A means of binding one or more *certified* public keys to a name. In the case of X.509, this is achieved by having *trusted certification authorities* issue digitally signed *certificates* each of which binds a name to a public key.

In addition, most PKI technologies have other features that allow users of a PKI to (for example) register, have their public keys certified and have their certificates revoked. However, such features are not used directly within protocols that use PKIs.

To model a PKI we take an abstract view of the functionality available to a protocol using the PKI. This requires three extensions to the spi-calculus.

1. The introduction of *PKI-Names*. These are normal π-calculus names that denote users of the PKI. Each PKI user will have a name and an associated key pair.
2. The ability to specify that a particular process is *acting on behalf* of a PKI user. The identification of the user associated with a process is used to enforce access to private keys (i.e., only a process acting of behalf of a user a can obtain a's private key) and as a mechanism for determining what information a user knows at the end of a protocol run.
3. The introduction of primitives that allow a process to obtain the private key, public key or certified public key of a PKI user. These primitives hide the details of how private keys are stored and how a PKI can be used to obtain a certified public key.

Our treatment of PKIs does not deal directly with *validity dates*, *certificate revocation*, *certificate paths* or the possibility that a user may have *multiple key-pairs*. Handling validity dates, certificate revocation and certificate paths is subsumed into the process of obtaining a certified key.

3.2 Syntax

The syntax of terms is the same as in the spi-calculus, but the syntax for processes is extended (see Figure 3) to include the explicit typing of names (see Figure 4) and the addition of primitives to handle public-key pairs *belonging* to PKI users.

$P, Q, R ::=$	
$\ldots$	
$(\nu\ n : T)P$	typed restriction (scope of n is P)
$\ldots$	
let $x =$ **public**(M) **in** P	bind the public key of user M to x in P
let $x =$ **certified**(M) **in** P	bind the *certified* public key of user M to x in P
let $x =$ **private**(M) **in** P	bind the private key of user M to x in P

Figure 3: SPIKY Syntax of processes

- The typed restriction $(\nu\ n : T)P$ introduces a new name n of type T that has an initial scope restricted to P. Note that type-checking is performed at run-time and variables remain untyped.

$T ::= \quad \mathit{Nonce} \mid \mathit{Key} \mid \mathit{KeyPair} \mid \mathit{Channel}$

Figure 4: SPIKY Syntax of types

- In the process **let** $x =$ **public**(M) **in** P, if M is not the name of a PKI-user then the process is stuck.

  If the process **let** $x =$ **public**(M) **in** P is *acting on behalf of the user* M and k_M^+ is M's public key, then k_M^+ is bound to x in P, i.e., the process behaves as $P[k_M^+/x]$[8].

  If the process **let** $x =$ **public**(M) **in** P is acting on behalf of a user N not equal to M, a public key (say k^+) is bound to x in P, i.e., the process behaves as $P[k^+/x]$. In this case the public key k^+ bound to x may be M's public key, another PKI-user's public key or some other public key. This process is used to model incorrect behaviour or protocol steps that avoid the overheads associated with public key validation.

- In the process **let** $x =$ **certified**(M) **in** P, if M is not the name of a PKI-user then the process is stuck.

  Otherwise for the process **let** $x =$ **certified**(M) **in** P, if M's public key is k_M^+, then k_M^+ is bound to x in P, i.e., the process behaves as $P[k_M^+/x]$.

  This process binds a *certified* public key to x by performing the necessary computations required by the PKI to validate the public key k_M^+.

- In the process **let** $x =$ **private**(M) **in** P, if M is not the name of a PKI-user then the process is stuck.

  If the process **let** $x =$ **private**(M) **in** P is acting on behalf of the user M and k_M^- is M's private key k_M^-, then k_M^- is bound to x in P, i.e., the process behaves as $P[k_M^-/x]$.

  If the process **let** $x =$ **private**(M) **in** P is acting on behalf of a user N not equal to M, then the process is stuck, i.e., a process can not acquire the private key of a user directly unless it is acting on behalf of that user.

We introduce a syntactic category of systems (see Figure 5) to specify processes acting on behalf of PKI-users. Informally, systems have the following meanings:

$E, F, G ::=$	
$E \mid F$	composition
$(\nu\ n : T)E$	extended typed restriction (scope of n is E)
$\{P\}^N$	The closed process P *acting* on behalf of user N

Figure 5: SPIKY Syntax of systems

[8]It should be noted that the subscript M in the name K_M is not part of the extended notation and is used solely to assist the reader.

- The system $E \mid F$ behaves as E and F running in parallel.
- The typed restriction $(\nu\ n : T)E$ introduces a new name n of type T (see Figure 4) that has an initial scope restricted to E.
- In the system $\{P\}^N$ the process P *acts on behalf of* the PKI user N. If N is not a name of type $User$, then $\{P\}^N$ is stuck.

Finally we introduce a syntactic category of protocols (see Figure 6) where a protocol is a pair consisting of a *state* $\phi : User \rightarrow KeyPair$ and a system E. A state ϕ maps distinct names of type $User$ to distinct names of type $KeyPair$. Informally, the protocol $(\phi : User \rightarrow KeyPair, E)$ executes the system E in the environment where ϕ specifies the key-pairs of each PKI user.

$$\begin{aligned} &Prot ::= \\ &\quad (\phi, E) \end{aligned}$$

Figure 6: SPIKY Syntax of protocols

We write **fn**(E) and **fv**(E) for the set of free names and variables, respectively, in the system E. A system is closed if it has no free variables. A protocol (ϕ, E) is a closed if $\mathbf{fv}(E) = \emptyset$ and $\mathbf{fn}(E) \subseteq \mathbf{dom}\ \phi \cup \mathbf{ran}\ \phi$.

3.3 Operational Semantics

To define the operational semantics of a protocol, we extend the reaction semantics of the spi-calculus [1, 2] with *reactions in the context of a state* $\xrightarrow{\phi}$, i.e., a reduction in which the key-pairs associated with names of type $User$ are obtained from ϕ. This semantics is defined in Figures 7, 8 and 9, where **type**(n) returns the type of the name n. Note that there are no rules for the new process primitives **public**, **certified** and **private** as reduction of these primitives can only take place within the context of a system. By convention, we treat $[(M_1, N_1)\ \mathbf{is}\ (M_2, N_2)]P$ as $[M_1\ \mathbf{is}\ M_2][N_1\ \mathbf{is}\ N_2]P$.

(Red Repl)	$!P$	$>$	$P \mid !P$	
(Red Match 1)	$[n\ \mathbf{is}\ n]P$	$>$	P	
(Red Match 2)	$[k^+\ \mathbf{is}\ k^+]P$	$>$	P	if $\mathbf{type}(k) = KeyPair$
(Red Match 3)	$[k^-\ \mathbf{is}\ k^-]P$	$>$	P	if $\mathbf{type}(k) = KeyPair$
(Red Let)	$\mathbf{let}\ (x_1, x_2) = (M_1, M_2)\ \mathbf{in}\ P$	$>$	$P[M_1/x_1][M_2/x_2]$	
(Red Decrypt Symm.)	$\mathbf{case}\ \{M\}_k\ \mathbf{of}\ \{x\}_k\ \mathbf{in}\ P$	$>$	$P[M/x]$	if $\mathbf{type}(k) = Key$
(Red Decrypt Asym.)	$\mathbf{case}\ \{\![M]\!\}_{k^+}\ \mathbf{of}\ \{\![x]\!\}_{k^-}\ \mathbf{in}\ P$	$>$	$P[M/x]$	if $\mathbf{type}(k) = KeyPair$
(Red Validate)	$\mathbf{case}\ [\!\{M\}\!]_{k^-}\ \mathbf{of}\ [\!\{x\}\!]_{k^+}\ \mathbf{in}\ P$	$>$	$P[M/x]$	if $\mathbf{type}(k) = KeyPair$

Figure 7: Extended Reduction Relation (>) on Processes

Figures 10, 11 and 12 define the operational semantics for systems.

(Struct Nil)	$P \mid \mathbf{0} \equiv P$		
(Struct Comm)	$P \mid Q \equiv Q \mid P$		
(Struct Assoc)	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$		
(Struct Switch)	$(\nu\ n : T_1)(\nu\ m : T_2)P \equiv (\nu\ m : T_2)(\nu\ n : T_1)P$		
(Struct Drop)	$(\nu\ n : T)\mathbf{0} \equiv \mathbf{0}$		
(Struct Extrusion)	$n \notin \mathbf{fn}(P)$	$\Rightarrow$	$(\nu\ n : T)(P \mid Q) \equiv P \mid (\nu\ n : T)Q$
(Struct Red)	$P > Q$	$\Rightarrow$	$P \equiv Q$
(Struct Refl)	$P \equiv P$		
(Struct Symm)	$P \equiv Q$	$\Rightarrow$	$Q \equiv P$
(Struct Trans)	$P \equiv Q \wedge Q \equiv R$	$\Rightarrow$	$P \equiv R$
(Struct Par)	$P \equiv P'$	$\Rightarrow$	$P \mid Q \equiv P' \mid Q$
(Struct Res)	$P \equiv Q$	$\Rightarrow$	$(\nu\ n : T)P \equiv (\nu\ n : T)Q$

Figure 8: Extented Structural Equivalence ($\equiv$) on Processes

(React Inter)	$\mathbf{type}(m) = \mathit{Channel}$	$\Rightarrow$	$\overline{m}\langle M\rangle.P \mid m(x).Q \longrightarrow P \mid Q[M/x]$
(React Struct)	$P \equiv P' \wedge P' \longrightarrow Q' \wedge Q' \equiv Q$	$\Rightarrow$	$P \longrightarrow Q$
(React Par)	$P \longrightarrow P'$	$\Rightarrow$	$P \mid Q \longrightarrow P' \mid Q$
(React Res)	$P \longrightarrow P'$	$\Rightarrow$	$(\nu\ n : T)P \longrightarrow (\nu\ n : T)P'$

Figure 9: Extended Reaction Relation ($\longrightarrow$) on Processes

(Red Red)	$\mathbf{type}(n) = \mathit{User} \wedge P > Q$	$\Rightarrow$	$\{P\}^n > \{Q\}^n$
(Red Comm)	$\mathbf{type}(n) = \mathit{User}$	$\Rightarrow$	$\{P \mid Q\}^n > \{P\}^n \mid \{Q\}^n$

Figure 10: Reduction Relation ($>$) on Systems

Most of the rules introduced in these definitions are straightforward extensions to the corresponding definitions for processes. However, a number of points should be noted:

- Given systems $\{P\}^n$ and $\{Q\}^m$ it may be necessary for P and Q to react with each other. To achieve this we introduce a new extrusion rule (*Struct Extrusion #1* in Figure 11) that allows restrictions to be moved in or out of systems and a new reaction rule (*Reaction Inter* in Figure 12) that permits input/output to occur between processes acting on behalf of different users.

- Figure 12 defines the reaction relation for the new process primitives **public**, **certified** and **private** acting on behalf of a user. *React Private* allows the private key of a user n to be retrieved by any process acting on behalf of n. *React Certified* allows any process acting on behalf of any user m to obtain a certified public key for any user n.

  The process primitive **public** is somewhat more complex and is captured by 3 rules. *React Public #1* allows a process acting of behalf of a user n to obtain n's public key. *React*

(Struct Nil)	$\mathbf{type}(n) = User$	$\Rightarrow$	$E \mid \{\mathbf{0}\}^n \equiv E$
(Struct Comm)			$E \mid F \equiv F \mid E$
(Struct Assoc)			$E \mid (F \mid G) \equiv (E \mid F) \mid G$
(Struct Switch)	$(\nu\ n : T_1)(\nu\ m : T_2)E \equiv (\nu\ m : T_2)(\nu\ n : T_1)E$		
(Struct Extrusion #1)	$\mathbf{type}(n) = User$	$\Rightarrow$	$(\nu\ m : T)\{P\}^n \equiv \{(\nu\ m : T)P\}^n$
(Struct Extrusion #2)	$n \notin \mathbf{fn}(E)$	$\Rightarrow$	$(\nu\ n : T)(E \mid F) \equiv E \mid (\nu\ n : T)F$
(Struct Red)	$E > F$	$\Rightarrow$	$E \equiv F$
(Struct Refl)			$E \equiv E$
(Struct Symm)	$E \equiv F$	$\Rightarrow$	$F \equiv E$
(Struct Trans)	$E \equiv F \wedge F \equiv G$	$\Rightarrow$	$E \equiv G$
(Struct Par)	$E \equiv E'$	$\Rightarrow$	$E \mid F \equiv E' \mid F$
(Struct Res)	$E \equiv E'$	$\Rightarrow$	$(\nu\ n : T)E \equiv (\nu\ n : T)E'$

Figure 11: Structural Equivalence ($\equiv$) on Systems

(React Inter)	$\mathbf{type}(m) = Channel \wedge \mathbf{type}(n_1) = User \wedge \mathbf{type}(n_2) = User$	$\Rightarrow$
	$\{\overline{m}\langle M\rangle.P\}^{n_1} \mid \{m(x).Q\}^{n_2} \xrightarrow{\phi} \{P\}^{n_1} \mid \{Q[M/x]\}^{n_2}$	
(React Struct)	$E \equiv E' \wedge E' \xrightarrow{\phi} F' \wedge F' \equiv F \quad \Rightarrow \quad E \xrightarrow{\phi} F$	
(React Par)	$E \xrightarrow{\phi} E' \quad \Rightarrow \quad E \mid F \xrightarrow{\phi} E' \mid F$	
(React Res)	$E \xrightarrow{\phi} E' \quad \Rightarrow \quad (\nu\ n : T)E \xrightarrow{\phi} (\nu\ n : T)E'$	
(React Private)	$n \in \mathbf{dom}\ \phi \quad \Rightarrow \quad \{\mathbf{let}\ x = \mathbf{private}(n)\ \mathbf{in}\ P\}^n \xrightarrow{\phi} \{P[\phi(n)^-/x]\}^n$	
(React Certified)	$n \in \mathbf{dom}\ \phi \wedge \mathbf{type}(m) = User \quad \Rightarrow$	
	$\{\mathbf{let}\ x = \mathbf{certified}(n)\ \mathbf{in}\ P\}^m \xrightarrow{\phi} \{P[\phi(n)^+/x]\}^m$	
(React Public #1)	$n \in \mathbf{dom}\ \phi \quad \Rightarrow \quad \{\mathbf{let}\ x = \mathbf{public}(n)\ \mathbf{in}\ P\}^n \xrightarrow{\phi} \{P[\phi(n)^+/x]\}^n$	
(React Public #2)	$n' \in \mathbf{dom}\ \phi \wedge \mathbf{type}(n) = User \wedge \mathbf{type}(m) = User \wedge m \neq n$	$\Rightarrow$
	$\{\mathbf{let}\ x = \mathbf{public}(n)\ \mathbf{in}\ P\}^m \xrightarrow{\phi} \{P[\phi(n')^+/x]\}^m$	
(React Public #3)	$\mathbf{type}(n) = User \wedge \mathbf{type}(m) = User \wedge k \notin \mathbf{fn}(P) \wedge m \neq n$	$\Rightarrow$
	$\{\mathbf{let}\ x = \mathbf{public}(n)\ \mathbf{in}\ P\}^m \xrightarrow{\phi} (\nu\ k : KeyPair)\{P[k^+/x]\}^m$	
(Struct Trans)	$E \equiv F \wedge F \equiv G \quad \Rightarrow \quad E \equiv G$	
(Struct Par)	$E \equiv E' \quad \Rightarrow \quad E \mid F \equiv E' \mid F$	
(Struct Res)	$E \equiv E' \quad \Rightarrow \quad (\nu\ n : T)E \equiv (\nu\ n : T)E'$	

Figure 12: Reaction Relation ($\xrightarrow{\phi}$) on Systems

Public #2 and *React Public #3* capture the possible reactions when a process acting on behalf of a user m attempts to obtain the public key of different user n. *React Public #2* allows the certified public key of some user n' (possibly equal to n) to be obtained and *React Public #3* allows some arbitrary public key to be obtained.

- The set of rules used to define the operational semantics of SPIKY are not optimal, but for clarity, redundant rules are presented. For example, since every process must act on behalf of a user, the React Inter rule for processes can be subsumed into the React Inter rule for systems.

To complete the operational semantics, Figure 13 defines the reaction relation for a closed protocol (ϕ, E).

$$\text{(React Protocol)} \quad E \xrightarrow{\phi} E' \quad \Rightarrow \quad (\phi, E) \longrightarrow (\phi, E')$$

Figure 13: Reaction Relation ($\longrightarrow$) on Protocols

4 Examples

In this section we present two examples; a simple authentication protocol and a more complex protocol that uses the **public** primitive. For reasons of space, we do not formalise notions of *authenticity*, but for each of our examples we give an informal justification for asserting that the protocol does indeed have an authenticity property.

Given a *claimant* B and *verifier* A, we consider B to be *authenticated* if A can establish that B has participated in the given instance of the protocol. Typically, we can show this by demonstrating that B's private key has been used to encrypt or decrypt some fresh nonce. It should be noted that this definition of authentication only ensures that a claimant has taken part in a protocol exchange. It does not guarantee that the protocol has not been subject to *manipulation* by an attacker.

A more complete approach to authentication would include the establishment of a shared secret key known by only A and B that was used for securing subsequent communications. This would then require analysis that only A and B have knowledge of the shared key.

4.1 A Simple Authentication Protocol

The following simple protocol allows two entities A and B to perform mutual authentication:

$$\begin{array}{llcl} (1.) & A \rightarrow B & : & A, N_A \\ (2.) & B \rightarrow A & : & B, N_B, \{\!|N_A|\!\}_{K_B^-} \\ (3.) & A \rightarrow B & : & \{\!|N_B|\!\}_{K_A^-} \end{array}$$

In step 1, A sends B its identity and a nonce, N_A. B signs this nonce and returns the signature together with its identity and a nonce N_B to A. Finally, A signs B's nonce and returns it to B. Of course, the entities A and B must validate signatures, handle public and private keys properly, etc..

In Figure 14 we present a system (abstraction) $SYST(a, b)$ that specifies communication between an initiator a and responder b using a shared channel ch. The behaviour of the initiator a is captured by the abstraction $INIT(a, b, ch)$ and that of the responder b by the abstraction $RESP(b, a, ch)$. Given these definitions, the protocol is defined by:

$$(\phi, SYST(A,B)) \tag{11}$$

where $A, B \in \mathbf{dom}(\phi)$.

$$\begin{array}{rcl}
INIT(a,b,ch) & \triangleq & (\nu\ n_a : Nonce)\overline{ch}\langle a, n_a\rangle.ch(b', n_b, sig). \\
& & \quad [b\ \mathbf{is}\ b']\ \mathbf{let}\ k_b = \mathbf{certified}(b)\ \mathbf{in} \\
& & \quad\quad [\ \{\!|sig, n_a|\!\}_{k_b}]\ \mathbf{let}\ k_a = \mathbf{private}(a)\ \mathbf{in}\ \overline{ch}\langle\{\!|n_b|\!\}_{k_a}\rangle.\mathbf{0} \\
\\
RESP(b,a,ch) & \triangleq & ch(a', n_a). \\
& & \quad [a\ \mathbf{is}\ a']\ \mathbf{let}\ k_b = \mathbf{private}(b)\ \mathbf{in} \\
& & \quad\quad (\nu\ n_b : Nonce)\overline{ch}\langle b, n_b, \{\!|n_a|\!\}_{k_b}\rangle.ch(sig). \\
& & \quad\quad\quad \mathbf{let}\ k_a = \mathbf{certified}(a)\ \mathbf{in}\ \ [\ \{\!|sig, n_b|\!\}_{k_a}].\mathbf{0} \\
\\
SYST(a,b) & \triangleq & (\nu\ ch : Channel)(\{INIT(a,b,ch)\}^a \mid \{RESP(b,a,ch)\}^b)
\end{array}$$

Figure 14: SPIKY definition of the simple authentication protocol

4.1.1 Authenticity

For our simple authentication protocol we consider the authenticity of A; a similar argument can be given for the authenticity of B.

For $RESP(B, A, ch)$ we observe that n_B is fresh, k_A is a certified copy of A's public key and $RESP(B, A, ch)$ only completes if sig is n_B signed with A's private key. These observations lead to a *partial correctness* result: if $RESP(B, A, ch)$ terminates, then A must have produced sig and therefore, must have participated in the protocol exchange. It should be noted that this result applies to $RESP$ and not the protocol exchange. Thus, authentication is independent of any manipulation that an attacker may perform on the protocol.

4.2 Mobile Authentication

In this example, we have two entities A and B that perform mutual authentication:

$$\begin{array}{lrcl}
(1.) & A \to B & : & A, N_A \\
(2.) & B \to A & : & B, N_B, \{\!|N_A|\!\}_{K_B^-} \\
(3.) & A \to S & : & \{\!|\{\!|N_B|\!\}_{K_A^-}, B, K_B^+|\!\}_{K_S^+} \\
(4.) & S \to B & : & \{\!|N_B|\!\}_{K_A^-}
\end{array}$$

However, entity A is assumed to execute on a small, mobile device that has insufficient capacity to obtain a certified copy of B's public key. Instead, A relies on a server S to ensure that the public key it has used is in fact B's public key. This is achieved in step 3 when A sends its signature of the nonce N_B, the name of the responder, B, and the key K_B^+ to S encrypted with S's public key. S checks this key and if it is B's public key, it *releases* A's signature of N_B. Note that:

1. The behaviour of B is the same as for the simple authentication protocol.
2. It is assumed that A can obtain a certified copy of S's public key. This may, for example, be achieved by having a copy of this key placed onto the mobile device during manufacture.

In Figure 15 we present a system (abstraction) $SYST(a, b, s)$ that specifies communication between an initiator a, a responder b and a server s using two channels ch and ch'. The extra channel ch' is used for communication from a to s. The behaviour of the initiator a is captured by the abstraction $INIT(a, b, s, ch, ch')$, that of the responder b by the abstraction $RESP(b, a, ch)$ and the server by $SERV(a, s, ch, ch')$. Given these definitions, the protocol is defined by:

$$(\phi, SYST(A, B, S)) \tag{12}$$

where $A, B, S \in \mathbf{dom}(\phi)$.

$$\begin{aligned}
INIT(a,b,s,ch,ch') \quad &\triangleq \quad (\nu\ n_a : Nonce)\overline{ch}\langle a, n_a\rangle.ch(b', n_b, sig). \\
&\qquad [b \textbf{ is } b'] \textbf{ let } k_b = \textbf{public}(b) \textbf{ in} \\
&\qquad\quad [\ \{\!|sig, n_a|\!\}_{k_b}] \textbf{ let } k_a = \textbf{private}(a) \textbf{ in} \\
&\qquad\quad\ \textbf{let } k_s = \textbf{certified}(s) \textbf{ in } \overline{ch'}\langle \{[\{\!|n_b|\!\}_{k_a}, b, k_b]\}_{k_s}\rangle.\mathbf{0} \\[1em]
RESP(b,a,ch) \quad &\triangleq \quad ch(a', n_a). \\
&\qquad [a \textbf{ is } a'] \textbf{ let } k_b = \textbf{private}(b) \textbf{ in} \\
&\qquad\quad (\nu\ n_b : Nonce)\overline{ch}\langle b, n_b, \{\!|n_a|\!\}_{k_b}\rangle.ch(sig). \\
&\qquad\quad\ \textbf{let } k_a = \textbf{certified}(a) \textbf{ in }\ [\ \{\!|sig, n_b|\!\}_{k_a}].\mathbf{0} \\[1em]
SERV(a,s,ch1,ch2) \quad &\triangleq \quad \textbf{let } k_s = \textbf{private}(s) \textbf{ in} \\
&\qquad ch'(c).\textbf{case } c \textbf{ of } \{[p]\}_{k_s} \textbf{ in} \\
&\qquad\quad \textbf{let } (sig, b, key) = p \textbf{ in} \\
&\qquad\quad\ \textbf{let } k_b = \textbf{certified}(b) \textbf{ in } [key \textbf{ is } k_b]\overline{ch}\langle sig\rangle.\mathbf{0} \\[1em]
SYST(a,b,s) \quad &\triangleq \quad (\nu\ ch : Channel)(\nu\ ch' : Channel) \\
&\qquad (\{INIT(a,b,s,ch,ch')\}^a\ | \\
&\qquad\ \{RESP(b,a,ch)\}^b\ | \\
&\qquad\ \{SERV(a,s,ch,ch')\}^s)
\end{aligned}$$

Figure 15: SPIKY definition of the mobile authentication protocol

4.2.1 Authenticity

For our mobile authentication protocol, the behaviour of B is the same for the simple authentication protocol. Thus, the authenticity of A follows the same argument as given in §(4.1.1) despite the use of S to relay $\{\!|N_B|\!\}_{K_A^-}$ to B.

In this case, however, establishing the authenticity of B requires a different form of argument, i.e., since $INIT(A, B, ch, ch')$ terminates before S has completed its work, authentication is dependent on both A and S terminating. For $INIT(A, B, ch, ch')$ we observe that n_A is fresh, sig is signed by a private key corresponding to k_B and k_S is S's public key. For $SERV(A, S, ch, ch')$ we observe that k_s is S's private key and it only terminates if k_B is is B's certified public key. These observations lead to a *partial correctness* result: if $INIT(A, B, ch, ch')$ and $SERV(A, S, ch, ch')$ terminate, then B must have produced sig and therefore, must have participated in the protocol exchange.

It should be noted that A can only be authenticated if $SERV(A, S, ch, ch')$ terminates, i.e., S will only send $\{\!|N_B|\!\}_{K_A^-}$ to B if it terminates.

5 Conclusions

In this paper we have presented an extension of the spi calculus that allows the modelling of PKI-based systems. The new notation incorporates primitives for the retrieval of (un)certified public and private keys of PKI users, and formalises the notion of process ownership.

Our notation lends itself to much clearer specifications of key usage in protocols and since processes act on behalf of users, it captures precisely what names are known by each participant in a protocol. We believe that our new notation will facilitate the formalisation of the authenticity and secrecy properties of protocols and that such an analysis can be built on previous experience in building static analyses that capture the term substitution property in nominal calculi [5, 4, 3].

References

[1] Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 414, University of Cambridge Computer Laboratory, January 1997.

[2] Martín Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the* 4th *ACM Conference on Computer and Communications Security*, pages 36–47, Zurich, Switzerland, April 1997. ACM Press.

[3] B. Aziz. *A Static Analysis Framework for Security Properties in Mobile and Cryptographic Systems*. PhD thesis, School of Computing, Dublin City University, Dublin, Ireland, 2003.

[4] B. Aziz, G.W. Hamilton, and D. Gray. A denotational approach to the static analysis of cryptographic processes. In *Proceedings of International Workshop on Software Verification and Validation*, volume 118, pages 19–36, Mumbai, India, December 2003. Electronic Notes in Theoretical Computer Science.

[5] Benjamin Aziz and Geoff Hamilton. A privacy analysis for the π-calculus: The denotational approach. In *Proceedings of the* 2nd *Workshop on the Specification, Analysis and Validation for Emerging Technologies*, number 94 in Datalogiske Skrifter, Copenhagen, Denmark, July 2002. Roskilde University.

[6] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proceedings of the* 6th *International Conference in Parallel Computing Technologies*, volume 2127 of *Lecture Notes in Computer Science*, pages 27–41, Novosibirsk, Russia, September 2001. Springer Verlag.

[7] Andrew Gordon. Nominal calculi for security and mobility. In D. Volpano, C. Irvine, and G. Smith, editors, *Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code*, pages 10–14, Monterey, California, USA, 1997. US Naval Postgraduate School.

[8] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[9] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, October 1996.

[10] Robin Milner. *Communicating and Mobile Systems: The Pi-Calculus.* Cambridge University Press, 1999.

[11] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes (parts I & II). *Information and Computation*, 100(1):1–77, September 1992.

[12] P.Y.A. Ryan and S.A. Schnieder. *Modelling and Analysis of Security Protocols.* Addison-Weslley, 2001.

[13] Davide Sangiorgi and David Walker. *The Pi-Calculus - A Theory of Mobile Processes.* Cambridge University Press, Cambridge, UK, 2001.

[14] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1937.

[15] CCITT Rec. X.500. ISO/IEC 9594,1–9:1994 information technology - open systems interconnection - the directory, 1994.

[16] CCITT Rec. X.509. ISO/IEC 9594–8:1994 information technology - open systems interconnection - the directory: Authentication framework, 1994.