Modelling and Analysis of PKI-based Systems Using Process Calculi

Benjamin Aziz[†] Geoff Hamilton[‡]

[†]Department of Computing, Imperial College London,
180 Queen's Gate, London, SW7 2RH, UK
‡School of Computing, Dublin City University, Dublin 9, Ireland

Abstract

In this technical report, we present a process algebra aimed at modelling PKI-based systems. The new language, SPIKY, extends the spi-calculus by adding primitives for the retrieval of certified/uncertified public keys as well as private keys belonging to users of the PKI-based system. SPIKY also formalises the notion of process ownership by PKI users, which is necessary in controlling the semantics of the key retrieval capabilities. We also construct a static analysis for SPIKY that captures the property of term substitutions resulting from message-passing and PKI/cryptographic operations. This analysis is shown to be safe and computable. Finally, we use the analysis to define the term secrecy and peer participation properties for a couple of examples of authentication 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 on security protocols 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, in the following 2-step protocol between agents A and B:

$$\begin{array}{lll} (1.)\ A \rightarrow B & : & A, N_A \\ (2.)\ B \rightarrow A & : & B, N_B, \{N_A\}_{K_{AB}} \end{array}$$

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.

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 [21] and nominal calculi [17] based on the π-calculus [27, 28, 33] 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 [1, 32].

While process algebras are normally computationally complete, i.e., they can be used to specify behaviour that is Turing-computable [35], 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 secret keys should be handled by an entity. In addition, there may be behaviour that is too complex to be captured succinctly in some algebra or which is not central to the behaviour of the protocol. 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 [1] 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 technical report we explore a simple extension to the spi-calculus [1] called SPIKY, which allows us to specify protocols that use *Public Key Infrastructures* (PKIs). More specifically, this extension allows us to formalize the binding of public keys to their owners and to give a more complete formal account of how PKI-based protocols behave. PKIs such as X.509 [36] 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 extension is intended to be independent of any particular PKI technology, so we use an abstract view of the functionality of PKIs in general.

We also construct a non-uniform static analysis for SPIKY that captures the property of term substitutions occurring in PKI systems [5] as a result of agents exchanging messages and performing PKI-related and cryptographic operations over those messages. In particular, the analysis captures PKI users sending and obtaining the substituted terms. Based on this information, it is possible to formalise security properties like for example, whether a user is capable of learning a term, and whether (un)certified public keys were used to arrive at the fact that B (respectively A) participated in the protocol.

The work presented in this report is an extension of previous works, [5, 19, 6]. In [19], we presented a previous version of the SPIKY language, its syntax and

structural operational semantics, and gave a couple of examples for simple and mobile authentication protocols. The version of SPIKY defined in [19] used a typing system for names, in order to distinguish cryptographic key pairs, channels, nonces and PKI users. In the current version, we have removed this classification except for PKI user names, which are taken as a mutually exclusive set. In [5], a static analysis for capturing name substitutions was presented. This analysis was used to further define certified and uncertified peer-entity participation properties.

Other related works in the area of the formal verification of PKI-based systems include [8, 20, 22, 23]. In [20], control flow analysis techniques [11, 30] based on flow logics [29] are used to validate the security properties of the SAML Single Sign-On Protocol [24, 25] within the LySA calculus [9]. In [22], model checking techniques are used to analyse access control properties of SPKI/SDSI name certificates [15] specified in the pushdown systems representation [12, 16]. In [8], a model for the analysis of trust-based properties in PKI systems is constructed based on a predicate logic similar to belief logics. Finally, [23] provides a validation analysis of X-509 certificates based on the HOL theorem prover [18].

The rest of the report is structured as follows. In Section 2, we discuss issues related to scope and ciphertext equality in nominal calculi. In Section 3, we introduce the syntax and structural operational semantics of the SPIKY language. In Section 4, we specify a couple of authentication protocols in the new language. In Section 5, we define a domain-theoretic model of the SPIKY language and define a denotational semantics. In Section 6, we define a non-standard semantics which captures the term-substitution property. In Section 7, we introduce an approximation which limits the number of new names generated in the semantics thus ensuring termination. In Section 8, we define the specification of Dolev-Yao's most general intruder. In Section 9, we define a couple of security properties on the results of the abstract semantics: the term secrecy and peer-entity participation properties. In Section 10, we analyse the two authentication protocols introduced earlier. Finally, in Section 11, we conclude the report and discuss future work.

2 On Scope and Ciphertext Equality

2.1 Scope

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 usual presence of replication and recursive abstractions makes it necessary to perform renaming dynamically during runtime, as a process evolves, if one is to obtain a clash-free semantics. For example, we can statically rename multiple occurrences of n in (1) and (2), but for (3), since there is an infinite number of occurrences of $(\nu n)P$, renaming must occur dynamically as the process evolves. One solution to this problem that we adopt in Section 5 onwards is to subscript occurrences of n with the number of the copy of the replicated process to which they belong, as n_1, n_2, etc.

2.2 Ciphertext Equality

Encryption schemes may either be *randomised* or *deterministic* [26]. 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. On the other hand, 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. 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], Abadi and Gordon specify that a match $[M \textit{ is } N]P$ 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 and Gordon do not give an explicit definition of what constitutes a match (although later works, such as [10], 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 depending on the particular encryption scheme being used. This makes it difficult to reason about 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[1], we will ignore this problem.

3 SPIKY

In this section, we define the syntax and structural operational semantics of SPIKY.

3.1 Syntax

The syntax of the SPIKY language is shown in Figure 1. This syntax consists of *terms*, *processes*, *systems* and *protocols*. The main building blocks of this syntax are terms, $L, M, N \in \mathcal{T}$. Terms are essentially composed from sets of names, $a, b, c, k, m, n \in \mathcal{N}$, variables, $v, x, y, z \in \mathcal{V}$ and PKI users (agents), $A, B, C, U \in \mathcal{AG}$. Additionally, a term may be a pair, (M, L), a symmetric ciphertext, $\{M\}_N$, a public-key ciphertext, $\{\!|M|\!\}_N$ and a digital signature, $[\!\{M\}\!]_N$. For convenience, we also refer to the private (public) component of a key pair as M^- (M^+)[2].

Processes, $P, Q, R \in \mathcal{P}$, are defined as follows. An output process, $\overline{M}\langle N\rangle.P$, is ready to emit N over channel M and continue as P. An input, $M(x).P$, is ready to input a message, L, over channel M and continue as $P[L/x]$. The parallel composition, $P \mid Q$, interleaves processes P and Q together. A restriction, $(\nu n)P$, creates a new name, n, and restricts its scope to P. A replicated process, $!P$, is capable of spawning infinitely many copies of P. Hence, replication is used to model infinite behaviour in SPIKY specifications. A match, $[M \ is \ N]P$, proceeds as P if M is the same as N, else it blocks. Due to the problems associated with matching ciphertexts and digital signatures as discussed in [19], we avoid any attempt to match these and restrict ourselves to name comparison. A null

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

[2] These components are defined more formally in the semantics of protocols in the next section.

L, M, N	::=		terms
		$a, b, c, k, m, n \in \mathcal{N}$	names
		$x, y, z, v, w \in \mathcal{V}$	variables
		$A, B, C, U \in \mathcal{AG}$	agents
		$\{M\}_N$	symmetric encryption
		$\{\!|M|\!\}_N$	public-key encryption
		$[\!\{M\}\!]_N$	digital signature
		(M, N)	pair
		M^+	public key component
		M^-	private key component
P, Q, R	::=		processes
		$\overline{M}\langle N\rangle.P$	output
		$M(x).P$	input
		$P \mid Q$	parallel composition
		$(\nu n)P$	restriction
		$!P$	replication
		$[M \ is \ N]P$	match
		$\mathbf{0}$	null
		$let\ (x, y) = M\ in\ P$	pair splitting
		$case\ L\ of\ \{x\}_N\ in\ P$	symmetric decryption
		$case\ L\ of\ \{\!|x|\!\}_N\ in\ P$	public-key decryption
		$case\ L\ of\ [\!\{x\}\!]_N\ in\ P$	signature with recovery validation
		$A(M)$	abstraction instantiation, where $A \stackrel{\mathtt{def}}{=} (x)P$
		$let\ x = \mathbf{private}(M)\ in\ P$	private key retrieval
		$let\ x = \mathbf{public}(M)\ in\ P$	public key retrieval
		$let\ x = \mathbf{certified}(M)\ in\ P$	certified public key retrieval
E, F, G	::=		systems
		$E \mid F$	parallel composition
		$(\nu n)E$	restriction
		$\lceil P \rceil^N$	process ownership
$Prot$	::=		protocols
		(θ, E)	(PKI state, system) pair

Figure 1: The syntax of the SPIKY language.

process, $\mathbf{0}$, cannot evolve any further. Pair splitting, $let\ (x, y) = M\ in\ P$, attempts to split a pair, M, into its first and second elements. It then assigns the first element to x and the second to y. Both x and y are bound variables. A symmetric decryption process, $case\ L\ of\ \{x\}_N\ in\ P$, attempts to decrypt L using the key, N. If this is successful, the result instantiates x, whose scope is P, otherwise, the process blocks. Similarly, $case\ L\ of\ \{\!|x|\!\}_N\ in\ P$, attempts to decrypt L using the public key, N, and the result instantiates x, which is a bound variable. The signature with recovery validation process, $case\ L\ of\ [\!\{x\}\!]_N\ in\ P$, behaves as $P[L/x]$ only if L is the signature $[\{M\}]_{k^-}$ and where N must be the public component of k. The abstraction instantiation, $A(M)$, assumes that a corresponding non-recursive definition, $A \stackrel{\mathtt{def}}{=} (x)P$, where M replaces x in P whenever the instantiation is called. Finally, the PKI operations, $let\ x = \mathbf{private}(M)\ in\ P$, $let\ x = \mathbf{public}(M)\ in\ P$ and $let\ x = \mathbf{certified}(M)\ in\ P$, attempt to perform

the PKI operations of retrieving private, uncertified public and certified public key components, respectively, of a PKI agent, M. The result of the operation is bound to x whose scope is P. Intuitively, the difference between $\mathbf{certified}(M)$ and $\mathbf{public}(M)$ is that the former corresponds to the PKI user effectively obtaining a valid public key of M at real time by validating the certification path all the way up to a root authority trusted by the user. This will insure that, for example, the key has not been revoked recently. On the other hand, the latter does not necessarily perform this validation at real time, i.e. it may return on an old copy of the public key of M without checking any recent revocation lists. In general, the use of $\mathbf{public}(M)$ is needed to account for any functional uncertainties in the PKI. The success or failure of the PKI operations depends on the ownership of the process.

Systems, $E, F \in \mathcal{E}$, are defined in order to model processes that run on behalf of PKI users, written as $\lceil P \rceil^M$, where M is an agent name. Hence, M may be regarded as the *owner* of P. Like processes, systems may be composed in parallel, $E \mid F$, and may have a name restriction, $(\nu n)E$. Finally, protocols, $Prot \in \mathcal{PR}$, are defined as pairs whose first element is a *PKI state*, $\theta : \mathcal{AG} \rightarrow \mathcal{N}$, mapping a PKI user to its key pair name. Intuitively, a protocol expresses the fact that every system, E, must be running over some PKI state, θ, in order for E to use its PKI operations.

In the rest of the report, we assume the reader to be familiar with the standard notions of α-conversion, term substitution and free/bound names and free/bound variables (referred to as $\mathbf{fn}(), \mathbf{bn}(), \mathbf{fv}(), \mathbf{bv}()$, respectively). The name, n, is bound in $(\nu n)P$ and in $(\nu n)E$. Otherwise, n is a free name. On the other hand, the variables, x and y, are bound in $M(x).P$, $A \stackrel{\mathsf{def}}{=} (x)P$, *let* $(x, y) = M$ *in* P, *case* L *of* $\{x\}_N$ *in* P, *case* L *of* $\{\![x]\!\}_N$ *in* P, *case* L *of* $[\![x]\!]_N$ *in* P, *let* $x =$ $\mathbf{private}(M)$ *in* P, *let* $x =$ $\mathbf{public}(M)$ *in* P and finally in *let* $x =$ $\mathbf{certified}(M)$ *in* P. Otherwise, x and y are free variables. In general, we write, $\mathbf{n}(e) = \mathbf{fn}(e) \cup \mathbf{bn}(e)$, to denote the set of all names of some entity, e (term, process, system or protocol). We also write $\mathbf{term}(e)$ to refer to the set of all terms appearing in e. Finally, we only deal with *normal protocols*.

Definition 1 *A protocol, Prot, is said to be normal if the following holds:*

- *The protocol is closed, i.e.* $\mathbf{fv}(Prot) = \{\}$,
- *There are no occurrences of homonymous bound names or homonymous bound variables in Prot, i.e.* $\forall x, y \in \mathbf{bv}(Prot), n, m \in \mathbf{bn}(Prot) : x \neq y \wedge n \neq m$,
- $\mathbf{bv}(Prot) \cap \mathbf{bn}(Prot) \cap \mathbf{fn}(Prot) = \{\}$.

3.2 Structural Operational Semantics

We define in this section a structural operational semantics for SPIKY. In general, this semantics is based on three main relations: the *reduction*, *structural*

congruence and *reaction* relations, each defined for the cases of processes, systems and protocols. First, we define the semantics of processes as in Figure 2. These rules are standard, however, note that in (RedMatch), the rule is only defined for names, n, as expected. Next, we define the semantics of systems

(RedRepl)	$!P$	$>$	$P \mid !P$
(RedMatch)	$[n \ is \ n]P$	$>$	P
(RedLet)	$let\ (x_1, x_2) = (M_1, M_2)\ in\ P$	$>$	$P[M_1/x_1][M_2/x_2]$
(RedDecryptSymm)	$case\ \{M\}_k\ of\ \{x\}_k\ in\ P$	$>$	$P[M/x]$
(RedDecryptAsym)	$case\ \{\!|M|\!\}_{k+}\ of\ \{\!|x|\!\}_{k-}\ in\ P$	$>$	$P[M/x]$
(RedRecValidate)	$case\ [\!\{M\}\!]_{k-}\ of\ [\!\{x\}\!]_{k+}\ in\ P$	$>$	$P[M/x]$
(RedAbstraction)	$A(M)$ where, $A \stackrel{\text{def}}{=} (x)P$	$>$	$P[M/x]$
(StructNil)	$P \mid \mathbf{0} \equiv P$		
(StructComm)	$P \mid Q \equiv Q \mid P$		
(StructAssoc)	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$		
(StructSwitch)	$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$		
(StructDrop)	$(\nu n)\mathbf{0} \equiv \mathbf{0}$		
(StructExtrusion)	$n \notin \mathbf{fn}(P) \ \Rightarrow\ (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$		
(StructRed)	$P > Q$	$\Rightarrow$	$P \equiv Q$
(StructRefl)	$P \equiv P$		
(StructSymm)	$P \equiv Q$	$\Rightarrow$	$Q \equiv P$
(StructTrans)	$P \equiv Q \ \wedge\ Q \equiv R$	$\Rightarrow$	$P \equiv R$
(StructPar)	$P \equiv P'$	$\Rightarrow$	$P \mid Q \equiv P' \mid Q$
(StructRes)	$P \equiv Q$	$\Rightarrow$	$(\nu n)P \equiv (\nu n)Q$
(ReactInter)	$\overline{m}\langle M\rangle.P \mid m(x).Q \longrightarrow P \mid Q[M/x]$		
(ReactStruct)	$P \equiv P' \wedge P' \longrightarrow Q' \wedge Q' \equiv Q$	$\Rightarrow$	$P \longrightarrow Q$
(ReactPar)	$P \longrightarrow P'$	$\Rightarrow$	$P \mid Q \longrightarrow P' \mid Q$
(ReactRes)	$P \longrightarrow P'$	$\Rightarrow$	$(\nu n)P \longrightarrow (\nu n)P'$

Figure 2: Rules of the $>$, $\equiv$ and $\longrightarrow$ relations on processes

as in Figure 3. Given systems $\lceil P \rceil^A$ and $\lceil Q \rceil^B$ it may be necessary for P and Q to react with each other. To achieve this we introduce a new extrusion rule (StructExtrusion) that allows restrictions to be moved in or out of systems, and a new reaction rule (ReactionInter) that permits input/output to occur between processes acting on behalf of different users. Finally, we define the semantics of protocols as in Figure 4. The reduction relation defines rules for all the PKI primitives. Rule (PRedPrivate) allows the private key of a user A to be retrieved by a process acting on behalf of A, whereas rule (PRedCertified) allows any process acting on behalf of any user B to obtain the (certified) public key for any other user, A. The process primitive **public** is somewhat more complex and is captured by three rules. Rule (PRedPublic#1) allows a process acting on behalf of a user, A, to obtain A's public key (this is similar to performing **certified**(A)). Rules (PRedPublic#2) and (PRedPublic#3) capture the possible reactions when a process acting on behalf of a user, B, attempts to obtain the public key of a different user, A. In this case, the result of executing **public**(A) may not yield the desired result (i.e. A's public key). Rule (RedPublic#2) says

(SySRedRed)	$P > Q$	$\Rightarrow$	$\lceil P \rceil^A >_E \lceil Q \rceil^A$
(SySRedComm)	$\lceil P \mid Q \rceil^A >_E \lceil P \rceil^A \mid \lceil Q \rceil^A$		
(SySStructProc)	$P \equiv Q$	$\Rightarrow$	$\lceil P \rceil^A \equiv_E \lceil Q \rceil^A$
(SySStructNil)	$E \mid \lceil \mathbf{0} \rceil^A \equiv_E E$		
(SySStructComm)	$E \mid F \equiv_E F \mid E$		
(SySStructAssoc)	$E \mid (F \mid G) \equiv_E (E \mid F) \mid G$		
(SySStructSwitch)	$(\nu n)(\nu m)E \equiv_E (\nu m)(\nu n)E$		
(SySStructExtr#1)	$(\nu m)\lceil P \rceil^A \equiv_E \lceil (\nu m)P \rceil^A$		
(SySStructExtr#2)	$n \notin \mathbf{fn}(E)$	$\Rightarrow$	$(\nu n)(E \mid F) \equiv_E E \mid (\nu n)F$
(SySStructRed)	$E >_E F$	$\Rightarrow$	$E \equiv_E F$
(SySStructRefl)	$E \equiv_E E$		
(SySStructSymm)	$E \equiv_E F$	$\Rightarrow$	$F \equiv_E E$
(SySStructTrans)	$E \equiv_E F \wedge F \equiv_E G$	$\Rightarrow$	$E \equiv_E G$
(SySStructPar)	$E \equiv_E E'$	$\Rightarrow$	$E \mid F \equiv_E E' \mid F$
(SySStructRes)	$E \equiv_E E'$	$\Rightarrow$	$(\nu n)E \equiv_E (\nu n)E'$
(SySReactInter)	$\lceil \overline{m}\langle M \rangle.P \rceil^A \mid \lceil m(x).Q \rceil^B \longrightarrow_E \lceil P \rceil^A \mid \lceil Q[M/x] \rceil^B$		
(SySReactProc)	$P \longrightarrow Q \Rightarrow \lceil P \rceil^A \longrightarrow_E \lceil Q \rceil^A$		
(SySReactStruct)	$E \equiv_E E' \wedge E' \longrightarrow_E F' \wedge F' \equiv_E F \Rightarrow E \longrightarrow_E F$		
(SySReactPar)	$E \longrightarrow_E E' \Rightarrow E \mid F \longrightarrow_E E' \mid F$		
(SySReactRes)	$E \longrightarrow_E E' \Rightarrow (\nu n)E \longrightarrow_E (\nu n)E'$		

Figure 3: Rules of the $>_E$, $\equiv_E$ and $\longrightarrow_E$ relations on systems

(PRedRed)	$E >_E F$	$\Rightarrow$	$(\theta, E) >_{Prot} (\theta, F)$
(PRedPrivate)	$A \in \mathbf{dom}\ \theta \Rightarrow$ $(\theta, \lceil let\ x = \mathbf{private}(A)\ in\ P \rceil^A) >_{Prot} (\theta, \lceil P[\theta(A)^-/x] \rceil^A)$		
(PRedCertified)	$A \in \mathbf{dom}\ \theta \Rightarrow$ $(\theta, \lceil let\ x = \mathbf{certified}(A)\ in\ P \rceil^B) >_{Prot} (\theta, \lceil P[\theta(A)^+/x] \rceil^B)$		
(PRedPublic#1)	$A \in \mathbf{dom}\ \theta \Rightarrow$ $(\theta, \lceil let\ x = \mathbf{public}(A)\ in\ P \rceil^A) >_{Prot} (\theta, \lceil P[\theta(A)^+/x] \rceil^A)$		
(PRedPublic#2)	$C \in \mathbf{dom}\ \theta \wedge B \neq A \Rightarrow$ $(\theta, \lceil let\ x = \mathbf{public}(A)\ in\ P \rceil^B) >_{Prot} (\theta, \lceil P[\theta(C)^+/x] \rceil^B)$		
(PRedPublic#3)	$k \notin \mathbf{fn}(P) \wedge B \neq A \Rightarrow$ $(\theta, \lceil let\ x = \mathbf{public}(A)\ in\ P \rceil^B) >_{Prot} (\theta, (\nu k)\lceil P[k^+/x] \rceil^B)$		
(PStructSys)	$E \equiv_E F$	$\Rightarrow$	$(\theta, E) \equiv_{Prot} (\theta, F)$
(PStructRed)	$Prot_1 >_{Prot} Prot_2$	$\Rightarrow$	$Prot_1 \equiv_{Prot} Prot_2$
(PStructRefl)	$Prot \equiv_{Prot} Prot$		
(PStructSymm)	$Prot \equiv_{Prot} Prot'$	$\Rightarrow$	$Prot' \equiv_{Prot} Prot$
(PStructTrans)	$Prot \equiv_{Prot} Prot' \wedge Prot' \equiv_{Prot} Prot''$	$\Rightarrow$	$Prot \equiv_{Prot} Prot''$
(PReactSys)	$E \longrightarrow_E F$	$\Rightarrow$	$(\theta, E) \longrightarrow_{Prot} (\theta, F)$
(PReactStruct)	$Prot_1 \equiv_{Prot} Prot_1' \wedge Prot_1' \longrightarrow_{Prot} Prot_2' \wedge Prot_2' \equiv_{Prot} Prot_2 \Rightarrow$ $Prot_1 \longrightarrow_{Prot} Prot_2$		

Figure 4: Rules of the $>_{Prot}$, $\equiv_{Prot}$ and $\longrightarrow_{Prot}$ relations on protocols

that executing $\mathbf{public}(A)$ by B may return the public key of any of the PKI users, C, currently registered in $\mathbf{dom}\ \theta$ (where C may or may not be A). Rule (PRedPublic#3) states that the returned result of the above operation may as

well be the public component of a fresh key pair, not belonging to any of θ's current registered users (this may be thought of as being a revoked key pair that is no more held in θ, or a key pair that was corrupted by noise while being retrieved).

4 Examples

We consider here a couple of examples of public-key authentication protocols [19], in order to demonstrate the use of SPIKY as a specification language. Sometimes, for the sake of simplicity, we write the pair $(M,(N,L))$ as (M,N,L) and we assume that $c(x,y,z).P$ stands for $c(u).let\ (x,u') = u\ in\ let\ (y,z) = u'\ in\ P$.

4.1 A Simple Authentication Protocol

The first protocol establishes mutual authentication between two agents, A and B:

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

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 5 we present a system (abstraction) $SYST$ that specifies communication between an initiator, A, and a responder, B, using a free channel, ch. The behaviour of

$$\begin{array}{lcl} INIT & \triangleq & (xa, xb, xch) \\ & & (\nu n_a)\overline{xch}\langle xa, n_a\rangle.ch(xb', xn_b, xsig). \\ & & \quad [xb\ is\ xb'] let\ xk_b = \mathbf{certified}(xb)\ in\ \underline{case}\ xsig\ of\ [\![x]\!]_{xk_b}\ in \\ & & \quad\quad [x\ is\ n_a]\ let\ xk_a = \mathbf{private}(xa)\ in\ \overline{xch}\langle [\![xn_b]\!]_{xk_a}\rangle.\mathbf{0} \\ & & \\ RESP & \triangleq & (yb, ych) \\ & & ych(ya, yn_a).let\ yk_b = \mathbf{private}(yb)\ in \\ & & \quad (\nu n_b)\overline{ych}\langle yb, yn_b, [\![yn_a]\!]_{yk_b}\rangle.ych(ysig). \\ & & \quad\quad let\ yk_a = \mathbf{certified}(ya)\ in\ \ case\ ysig\ of\ [\![y]\!]_{yk_a}\ in\ [y\ is\ yn_b]\mathbf{0} \\ & & \\ SYST & \triangleq & \lceil INIT(A,B,ch)\rceil^A \mid \lceil RESP(B,ch)\rceil^B \end{array}$$

Figure 5: SPIKY definition of the simple authentication protocol

the initiator is captured by the abstraction $INIT$ and that of the responder by the abstraction $RESP$. The protocol as a whole is defined for some PKI state, θ, as $(\theta, SYST)$.

4.2 A Mobile Authentication Protocol

We have again the two agents, A and B, trying to establish mutual authentication:

$$\begin{array}{lll} (1.) & A \rightarrow B & : \; A, N_A \\ (2.) & B \rightarrow A & : \; B, N_B, [\![\{N_A\}]\!]_{K_B{}^-} \\ (3.) & A \rightarrow S & : \; \{[\![[\![\{N_B\}]\!]_{K_A{}^-}, B, K_B{}^+]\!]\}_{K_S{}^+} \\ (4.) & S \rightarrow B & : \; [\![\{N_B\}]\!]_{K_A{}^-} \end{array}$$

However, in this case, 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 trusted server, S, to ensure that the copy of the public key it has just used to authenticate B is *indeed* 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 specified in the previous section.

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 6 we present a system, $SYST$, that specifies communication between an initiator A, a responder B and a server S using channels, ch and ch'.

The extra channel ch' is used for communications from A to S. The behaviour of the initiator is captured by the abstraction $INIT$, the responder by the abstraction $RESP$ and the server by the abstraction $SERV$. The protocol is defined for a particular instance of PKI state, θ, as $(\theta, SYST)$.

5 A Domain-Theoretic Model

In this section, we define a domain-theoretic semantics for the SPIKY language that is based on the model of processes originally defined by Stark [34] for the π-calculus and that was further extended for the case of the spi-calculus in [7] to deal with cryptographic processes. Our new model is based on the following

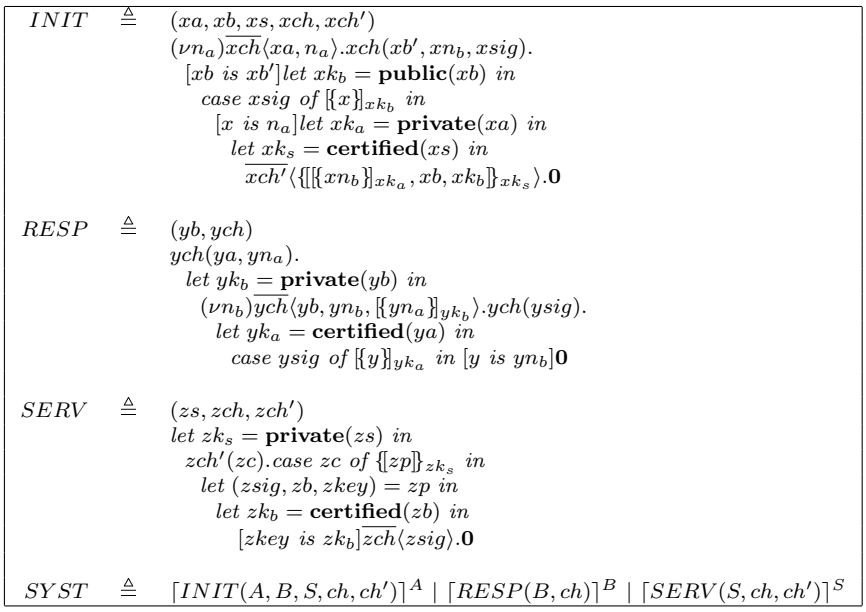

Figure 6: SPIKY definition of the mobile authentication protocol.

predomain equations, which describe what a closed process can do in SPIKY:

$$Spiky \cong 1 + \mathbb{P}(Spiky_\perp + In + Out) \quad (4)$$

$$In \cong N \times (T \to Spiky_\perp) \quad (5)$$

$$Out \cong N \times (T \times Spiky_\perp + N \to \ldots N \to (T \times Spiky_\perp)) \quad (6)$$

$$T \cong AG + N + Sec + Pub + Sig + Pair \quad (7)$$

$$Sec \cong T \times N \quad (8)$$

$$Pub \cong T \times N \quad (9)$$

$$Sig \cong T \times N \quad (10)$$

$$Pair \cong T \times T \quad (11)$$

Where $Spiky_\perp$ is the domain of processes, In and Out are the predomains of input and output actions, respectively. Input actions are modelled as pairs; a name, N (the channel), and a function, $T \to Spiky_\perp$, that can be instantiated with a term, T, yielding a process in $Spiky_\perp$. Output actions are divided into free and bound output actions. These are pairs consisting of the channel, N, and either another pair, $T \times Spiky_\perp$, denoting the message, T, and the residue $Spiky_\perp$ (free outputs), or composed functions, $N \to \ldots N \to (T \times Spiky_\perp)$, that introduce new names to the message, T, and the residue, $Spiky_\perp$ (bound outputs). $\mathbb{P}(-)$ is Plotkin's powerdomain [31] applied to the disjoint union of input, output and silent actions (the latter represented by $Spiky_\perp$) to construct $Spiky$. The one-element predomain, 1, representing terminated (deadlocked)

processes is adjoined as in [2]. The flat predomain of closed terms, T, is defined as the disjoint union of the predomains of PKI users, AG, names, N, secret-key ciphers, Sec, public-key ciphers, Pub, digital signatures, Sig, and pairs, $Pair$. The predomains Sec, Pub and Sig are represented as pairs, $T \times N$, where the term, T, is encrypted/signed using the key, N. There is no predomain of variables since we only deal with closed terms.

In order to be able to define a denotational semantics for the SPIKY language, we need to define concrete elements of each of the (pre)domains of (4)-(11). These elements are defined in Figure 7, where $\mathcal{K}$ is the set underlying any (pre)domain. Clearly from the definition of Figure 7, the domain $Spiky_\perp$ is a

- *Elements of* AG : $U \in \mathcal{AG} \Rightarrow U \in \mathcal{K}(AG)$
- *Elements of* N : $a \in \mathcal{N} \Rightarrow a \in \mathcal{K}(N)$
- *Elements of* Sec : $k \in \mathcal{K}(N), t \in \mathcal{K}(T) \Rightarrow sec(t,k) \in \mathcal{K}(Sec)$
- *Elements of* Pub : $k \in \mathcal{K}(N), t \in \mathcal{K}(T) \Rightarrow pub(t,k) \in \mathcal{K}(Pub)$
- *Elements of* Sig : $k \in \mathcal{K}(N), t \in \mathcal{K}(T) \Rightarrow sig(t,k) \in \mathcal{K}(Sig)$
- *Elements of* $Pair$: $t \in \mathcal{K}(T), t' \in \mathcal{K}(T) \Rightarrow (t,t') \in \mathcal{K}(Pair)$
- *Elements of* T : $\mathcal{K}(T) = \mathcal{K}(AG) + \mathcal{K}(N) + \mathcal{K}(Sec) \cup \mathcal{K}(Pub) \cup \mathcal{K}(Sig) \cup \mathcal{K}(Pair)$
- *Elements of* In :
  $a \in \mathcal{K}(N), p \in \mathcal{K}(Spiky_\perp) \Rightarrow (a, \lambda x.p) \in \mathcal{K}(In)$
- *Elements of* Out :
  $a \in \mathcal{K}(N), t \in \mathcal{K}(T), p \in \mathcal{K}(Spiky_\perp) \Rightarrow (a,t,p) \in \mathcal{K}(Out)$
  $a \in \mathcal{K}(N), t \in \mathcal{K}(T), p \in \mathcal{K}(Spiky_\perp) \Rightarrow (a, \lambda n_1, \ldots, \lambda n_m.(t,p)) \in \mathcal{K}(Out)$
- *Elements of* $Spiky_\perp$:
  $\{\!|\perp|\!\} \in \mathcal{K}(Spiky_\perp)$
  $\emptyset \in \mathcal{K}(Spiky_\perp)$
  $p, q \in \mathcal{K}(Spiky_\perp) \Rightarrow p \uplus q \in \mathcal{K}(Spiky_\perp)$
  $p \in \mathcal{K}(Spiky_\perp) \Rightarrow \{\!|tau(p)|\!\} \in \mathcal{K}(Spiky_\perp)$
  $e \in \mathcal{K}(In) \Rightarrow \{\!|in(e)|\!\} \in \mathcal{K}(Spiky_\perp)$
  $e \in \mathcal{K}(Out) \Rightarrow \{\!|out(e)|\!\} \in \mathcal{K}(Spiky_\perp)$
  $x \in \mathcal{K}(N), p \in \mathcal{K}(Spiky_\perp) \Rightarrow new(\lambda x, p) \in \mathcal{K}(Spiky_\perp)$

Figure 7: Elements of AG, N, Sec, Pub, Sig, T, In, Out and $Spiky_\perp$.

multiset of semantic processes (ref. to similar treatments in [3, 7]). The definition of this multiset utilises the usual multiset operations such as the empty multiset, $\emptyset$, the singleton multiset, $\{\!|\,|\!\}$, and the union of multisets, $\uplus$. The $\emptyset$ operation denotes inactive processes and the $\{\!|\,|\!\}$ operation creates elements of $Spiky_\perp$ from single elements of input, output and silent actions. On the other hand, $\uplus$ is needed to capture non-determinism in the semantics of processes.

In addition to the above standard multiset operations, we also introduce a special operator, *new*, which is needed to interpret the effects of restricting a name to a process. These effects are formalised in the definition of *new* in Figure 8 over fully evaluated elements of $Spiky_\perp$. In general, *new* blocks any attempts to communicate over fresh non-extruded channels. It also turns a free output into a bounded output whenever the message of communication is restricted. In all other cases, *new* has no effect and it is simply distributed over $\uplus$ or passed on to the residual process.

The denotational semantics for the SPIKY language is given as a semantic

$$
\begin{array}{lcl}
new(\lambda n.\emptyset) & = & \emptyset \\
new(\lambda n.\{\!|\bot|\!\}) & = & \{\!|\bot|\!\} \\
new(\lambda n.\{\!|in(a,\lambda x.p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } a = n \\ \{\!|in(a,\lambda x.new(\lambda n.p))|\!\}, & \text{otherwise} \end{cases} \\
new(\lambda n.\{\!|out(a,t,p)|\!\}) & = & \begin{cases} \emptyset, & \text{if } a = n \\ \{\!|out(a,\lambda n.(t,p))|\!\}, & \text{if } n \in n(t) \text{ and } n \neq a \\ \{\!|out(a,t,new(\lambda n.p))|\!\}, & \text{otherwise} \end{cases} \\
new(\lambda n.\{\!|out(a,\lambda m_1 \ldots \lambda m_k.(t,p))|\!\}) & = & \\
\multicolumn{3}{l}{\begin{cases} \emptyset, & \text{if } a = n \\ \{\!|out(a,\lambda n.\lambda m_1 \ldots \lambda m_k.(t,p))|\!\}, & \text{if } n \in n(t) \text{ and } n \neq a \\ \{\!|out(a,\lambda m_1 \ldots \lambda m_k.(t,new(\lambda n.p)))|\!\}, & \text{otherwise} \end{cases}} \\
new(\lambda n.\{\!|tau(p)|\!\}) & = & \{\!|tau(new(\lambda n.p))|\!\} \\
new(\lambda n.(p_1 \uplus p_2)) & = & new(\lambda n.p_1) \uplus new(\lambda n.p_2)
\end{array}
$$

Figure 8: The concrete definition of *new* over elements $p \in Spiky_\perp$.

function, $\mathcal{S}(\![E]\!)\ \rho\ \phi_\mathcal{S}\ \theta \in Spiky_\perp$, defined by the set of rules of Figure 9. The θ environment is defined as the PKI state of some protocol, such that $\theta(U)^+$ is the public key of U and $\theta(U)^-$ is its private key. The multiset, ρ, is used to hold systems composed in parallel with the analysed system. Furthermore, rule ($\mathcal{R}0$) is used to interpret the contents of ρ. The environment, $\phi_\mathcal{S} : V \to T$, where V is the flat predomain of variables, captures any term substitutions that occur in the semantics. Note that initially, $\forall u \in V + N + AG : \phi_{\mathcal{S}0}(u) = u.$. The special function, $\varphi_\mathcal{S}$, returns the semantic value of a term:

$$
\forall \phi_\mathcal{S}, M : \varphi_\mathcal{S}(\phi_\mathcal{S}, M) = \begin{cases} \phi_\mathcal{S}(M), & \text{if } M \in (AG + N + V) \\ sec(\varphi_\mathcal{S}(\phi_\mathcal{S}, M'), \varphi_\mathcal{S}(\phi_\mathcal{S}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_\mathcal{S}(\phi_\mathcal{S}, M'), \varphi_\mathcal{S}(\phi_\mathcal{S}, N)), & \text{if } M = \{\!|M'|\!\}_N \\ sig(\varphi_\mathcal{S}(\phi_\mathcal{S}, M'), \varphi_\mathcal{S}(\phi_\mathcal{S}, N)), & \text{if } M = [\![M']\!]_N \\ (\varphi_\mathcal{S}(\phi_\mathcal{S}, N), \varphi_\mathcal{S}(\phi_\mathcal{S}, L)), & \text{if } M = (N, L) \end{cases}
$$

Rules ($\mathcal{S}0A$) and ($\mathcal{S}0B$) interpret parallelism and restriction between two systems by joining the parallel systems to ρ and using the *new* operator, respectively. Rules ($\mathcal{S}1$)–($\mathcal{S}15$) deal with process ownership by cases. Rule ($\mathcal{S}1$), deals with output actions taking into consideration any communications that may occur between the output channel and appropriate input channels guarding processes in ρ. The $\phi_\mathcal{S}$ is updated appropriately with the substituted semantic elements. Rule ($\mathcal{S}2$) deals with input functions leaving out communications since these are considered in ($\mathcal{S}1$). Rule ($\mathcal{S}3$) interprets directly parallel composition by the addition of the parallel subprocesses to ρ. In the semantics of [34], a different operator called *par* is defined to interpret the meaning of parallel process, which also takes care of communications between output and input guarded processes. However the use of this operator would complicate the definition of our abstract semantics later in Section 7. Rule ($\mathcal{S}4$) uses *new* to interpret the meaning of a restriction. Rule ($\mathcal{S}5$) interprets a replication, $\lceil !P \rceil^U$, as the least upper bound of the infinite poset $\mathcal{F}$. This least upper bound represents the least fixed point meaning of $!P$. Due to the fact that the seman-

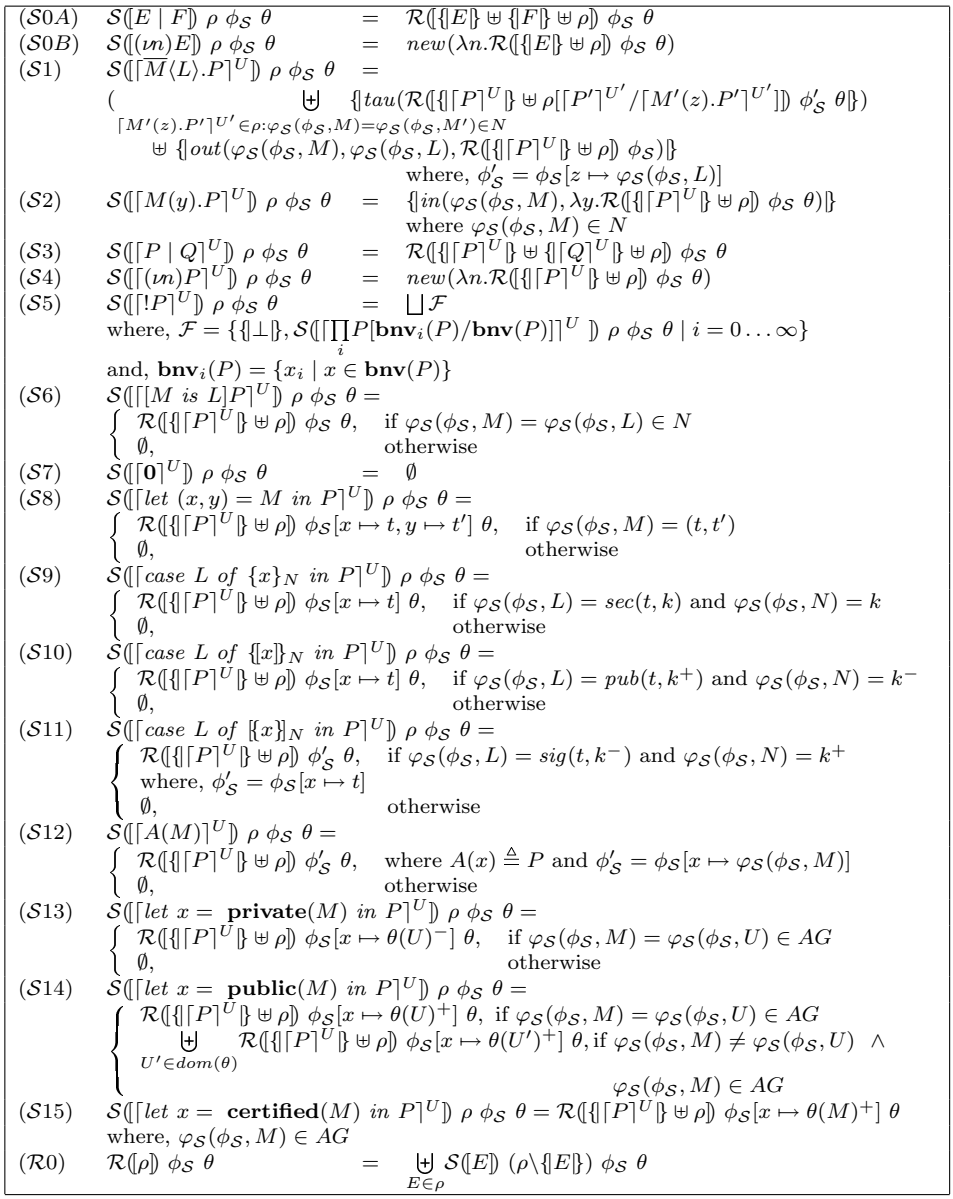

Figure 9: The standard denotational semantics of the SPIKY language.

tic domain, $Spiky_{\perp}$, is infinite, the calculation of this least fixed point may not terminate within finite limits. The rule also uses a labelling mechanism to rename all the bound variables and names, $\mathbf{bnv}(P)$, of the spawned processes by subscripting those variables and names with a number signifying process copy.

This maintains the normality requirement of Definition 1.

Rule ($\mathcal{S}6$) compares the meaning of two terms as given by $\varphi_{\mathcal{S}}$. As we mentioned earlier in Section 3, we restrict this comparison to names. Rule ($\mathcal{S}7$) interprets the meaning of a null system as the empty set mapping, $\emptyset$. Rule ($\mathcal{S}8$) splits the elements of a pair term. Rules ($\mathcal{S}9$)–($\mathcal{S}11$) deal with cryptographic systems for the decryption of symmetric and public-key ciphertexts and signatures with recovery and appendix validations. A residual system, $\lceil P \rceil^U$, signifying the success of the operation is added to ρ, else, if the operation fails, $\emptyset$ is returned instead. Rule ($\mathcal{S}12$) interprets the meaning of abstraction instantiations directly by adding the definition to ρ and updating $\phi_{\mathcal{S}}$ with the substituted term. Rules ($\mathcal{S}13$)–($\mathcal{S}15$) deal with PKI operations for retrieving private, uncertified and certified public keys. This is done using the PKI state, θ, and the user owning the system, U. The uncertified public key operation offers less guarantees (if the owner of the process requires other users' keys), therefore, it may return the public key of any PKI user, U', in $dom(\theta)$. On the other hand, the certified version is always guaranteed to return a valid public key, regardless of the owner's identity.

6 Non-Standard Semantics

We extend here the standard semantics of the previous section to a non-standard semantics that captures the property of term substitutions. For example, in:

$$(\theta, \lceil \overline{c}\langle k \rangle.P \rceil^U \mid \lceil c(x).Q \rceil^{U'})$$

We are interested in capturing the information that key, k, substitutes variable, x, and that this substitution happened due a communication *from* user U *to* user U'. In another example:

$$(\theta, \lceil let\ x = \mathbf{private}(U)\ in\ P \rceil^U)$$

Here, we are interested in capturing the fact that variable x will inevitably be instantiated with the private key of U, i.e. $\theta(U)^-$, and that this happens within a process owned by U. To be able to capture this kind of information, we need to define a new meaning for our systems in terms of a new special environment, $\phi_{\mathcal{E}} : V \to \wp(T \times AG \times AG)$, which maps each variable of a closed system to a set of triples representing semantic terms that may substitute the variable, and names of PKI users that instantiate and own that variable.

A non-standard semantic domain, $D_\perp = V \to \wp(T \times AG \times AG)$, can be constructed, ordered by subset inclusion as follows:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp : \phi_{\mathcal{E}1} \sqsubseteq_{D_\perp} \phi_{\mathcal{E}2} \Leftrightarrow \forall x \in V : \phi_{\mathcal{E}1}(x) \subseteq \phi_{\mathcal{E}2}(x)$$

with the bottom element, $\perp_{D_\perp}$, being the null environment, $\phi_{\mathcal{E}0}$, that maps each variable to the empty set. The union of environments operation, $\cup_\phi$, is defined as:

$$\forall \phi_{\mathcal{E}1}, \phi_{\mathcal{E}2} \in D_\perp, x \in V : (\phi_{\mathcal{E}1} \cup_\phi \phi_{\mathcal{E}2})(x) = \phi_{\mathcal{E}1}(x) \cup \phi_{\mathcal{E}2}(x)$$

The non-standard semantics of the SPIKY language is defined using the semantic function, $\mathcal{E}(\![E]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta \in D_{\perp}$, as illustrated in Figure 10. The definitions of

$$
\begin{array}{lll}
(\mathcal{E}0A) & \mathcal{E}(\![E \mid F]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \mathcal{R}(\![\{\!|E|\!\} \uplus \{\!|F|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}\ \theta \\
(\mathcal{E}0B) & \mathcal{E}(\![(\nu n)E]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \mathcal{R}(\![\{\!|E|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}\ \theta \\
(\mathcal{E}1) & \mathcal{E}(\![\ulcorner \overline{M}\langle L\rangle.P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \bigcup_{\phi} \quad \phi'_{\mathcal{E}}\ \cup_{\phi}\ \phi_{\mathcal{E}} \\
 & & \quad \ulcorner M'(z).P' \urcorner^{U'} \in \rho : \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M') \in N \\
 & \text{where, } \phi'_{\mathcal{E}} = \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho[\ulcorner P' \urcorner^{U'} / \ulcorner M'(z).P' \urcorner^{U'}]]\!)\ \phi_{\mathcal{E}}[z \mapsto \{(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L), U, U')\}]\ \theta & \\
(\mathcal{E}2) & \mathcal{E}(\![\ulcorner M(y).P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \phi_{\mathcal{E}} \\
(\mathcal{E}3) & \mathcal{E}(\![\ulcorner P \mid Q \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \{\!|\ulcorner Q \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}\ \theta \\
(\mathcal{E}4) & \mathcal{E}(\![\ulcorner (\nu n)P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}\ \theta \\
(\mathcal{E}5) & \mathcal{E}(\![\ulcorner !P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \bigsqcup \mathcal{F} \\
 & \text{where, } \mathcal{F} = \{\perp_{D_{\perp}}, \mathcal{E}(\![\ \ulcorner \prod_{i} P[\mathbf{bnv}_i(P)/\mathbf{bnv}(P)] \urcorner^{U}\ ]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta \mid\ i = 0 \ldots \infty\} & \\
 & \text{and, } \mathbf{bnv}_i(P) = \{x_i \mid x \in \mathbf{bnv}(P)\} & \\
(\mathcal{E}6) & \mathcal{E}(\![\ulcorner [M \textit{ is } L]P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) \in N \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}7) & \mathcal{E}(\![\ulcorner \mathbf{0} \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta & = \phi_{\mathcal{E}} \\
(\mathcal{E}8) & \mathcal{E}(\![\ulcorner \textit{let } (x,y) = M \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(t,U,U)\}, y \mapsto \{(t',U,U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = (t,t') \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}9) & \mathcal{E}(\![\ulcorner \textit{case } L \textit{ of } \{x\}_N \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = \mathit{sec}(t,k) \\ & \quad \text{and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}10) & \mathcal{E}(\![\ulcorner \textit{case } L \textit{ of } \{\!|x|\!\}_N \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = \mathit{pub}(t,k^{+}) \\ & \quad \text{and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^{-} \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}11) & \mathcal{E}(\![\ulcorner \textit{case } L \textit{ of } [\![x]\!]_N \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(t,U,U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L) = \mathit{sig}(t,k^{-}) \\ & \quad \text{and } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N) = k^{+} \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}12) & \mathcal{E}(\![\ulcorner A(M) \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M), U, U)\}]\ \theta, & \text{where } A(x) \triangleq P \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}13) & \mathcal{E}(\![\ulcorner \textit{let } x = \mathbf{private}(M) \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(\theta(U)^{-}, U, U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, U) \in AG \\ \phi_{\mathcal{E}}, & \text{otherwise} \end{cases} & \\
(\mathcal{E}14) & \mathcal{E}(\![\ulcorner \textit{let } x = \mathbf{public}(M) \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(\theta(U)^{+}, U, U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, U) \in AG \\ \quad \bigcup_{\phi} \quad \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(\theta(U')^{+}, U, U)\}]\ \theta, & \text{if } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \neq \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, U)\ \wedge \\ \quad U' \in dom(\theta) & \\ & \qquad \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in AG \end{cases} & \\
(\mathcal{E}15) & \mathcal{E}(\![\ulcorner \textit{let } x = \mathbf{certified}(M) \textit{ in } P \urcorner^{U}]\!)\ \rho\ \phi_{\mathcal{E}}\ \theta = & \\
 & \mathcal{R}(\![\{\!|\ulcorner P \urcorner^{U}|\!\} \uplus \rho]\!)\ \phi_{\mathcal{E}}[x \mapsto \{(\theta(M)^{+}, U, U)\}]\ \theta, \text{where, } \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) \in AG & \\
(\mathcal{R}0) & \mathcal{R}(\![\rho]\!)\ \phi_{\mathcal{E}}\ \theta & = \bigcup_{E \in \rho}{}_{\phi}\ \mathcal{E}(\![E]\!)\ (\rho \backslash \{\!|E|\!\})\ \phi_{\mathcal{E}}\ \theta
\end{array}
$$

Figure 10: The non-standard semantics of the SPIKY language.

ρ and θ are as in Section 5. The definition of the special function, $\varphi_{\mathcal{E}}$ allows for the meaning of a closed term to be computed under some $\phi_{\mathcal{E}}$:

$$\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M) = \begin{cases} t, & \text{if } M \in V \wedge \phi_{\mathcal{E}}(M) = \{(t, U, U')\} \\ M, & \text{if } M \in (N + AG) \\ sec(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{M'\}_N \\ pub(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = \{\![M']\!\}_N \\ sig(\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, N)), & \text{if } M = [\![M']\!]_N \\ (\varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'), \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, M'')), & \text{if } M = (M', M'') \end{cases}$$

Note that $\varphi_{\mathcal{E}}$ is only defined for the case of variables where the variable has only been instantiated with a singleton in $\phi_{\mathcal{E}}$. This is due to the fact that the non-standard semantics is a precise semantics, i.e. each variable is instantiated at most once per choice of control flow. This is clear from rule ($\mathcal{R}0$), where the $\phi_{\mathcal{E}}$ environment is passed unchanged from LHS to RHS of the rule. It is only at the top level (i.e. when computing $\cup_\phi$ in ($\mathcal{R}0$)) are the different instantiations of the same variable (due to non-determinism) combined together in a set.

The main difference in this semantics as compared to the standard semantics of the previous section is the fact that the meaning of a process is a $\phi_{\mathcal{E}}$ environment rather than an element of $Spiky_\perp$. Note again the difference in performing uncertified versus certified public key retrieval in rules ($\mathcal{E}14$) and ($\mathcal{E}15$), respectively. In the former case, the owner of a process may obtain any public key stored in θ when asking for some other user's public key without any guarantees as to the validity of the key-user binding (unless the owner asks for its own public key). In the latter case, this requirement is always guaranteed to return a public key that is validly bound to its user.

The following theorem establishes a correctness relation with respect to the standard denotational semantics of the previous section.

Theorem 1 (Correctness of the Non-Standard Semantics)
$\forall(\theta, E), \phi_{\mathcal{S}}, \phi_{\mathcal{E}}, x, \mathcal{S}[\![E]\!]\ \rho\ \phi_{\mathcal{S}}\ \theta = p(\mathcal{R}[\![\rho']\!]\ \phi'_{\mathcal{S}}\ \theta), \mathcal{E}[\![E]\!]\ \rho\ \phi_{\mathcal{E}}\ \theta = \mathcal{R}[\![\rho']\!]\ \phi'_{\mathcal{E}}\ \theta$*:*
$\exists U, U' : (\phi_{\mathcal{S}}(x), U, U') \in \phi_{\mathcal{E}}(x) \quad \Rightarrow \quad \exists U, U' : (\phi'_{\mathcal{S}}(x), U, U') \in \phi'_{\mathcal{E}}(x)$

Proof. The proof of the theorem is by structural induction on the structure of P. We only provide a proof sketch here of the most interesting cases. The base case is the case of the null protocol, $(\theta, \lceil \mathbf{0} \rceil^U)$ and this is satisfied from the antecedent of the theorem since neither rule ($\mathcal{S}7$) nor ($\mathcal{E}7$) change the initial $\phi_{\mathcal{S}}$ and $\phi_{\mathcal{E}}$ environments on LHS of the rule. The other interesting case is that of rules ($\mathcal{E}1$) and ($\mathcal{S}1$) where $\phi_{\mathcal{S}}$ and $\phi_{\mathcal{E}}$ are changed. In rule ($\mathcal{S}1$), $\phi_{\mathcal{S}}$ is changed on RHS to $\phi'_{\mathcal{S}} = \phi_{\mathcal{S}}[x \mapsto \varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L)]$. It is trivial to show from the antecedent that $\varphi_{\mathcal{S}}(\phi_{\mathcal{S}}, L) = \varphi_{\mathcal{E}}(\phi_{\mathcal{E}}, L)$, which leads to the changing of $\phi_{\mathcal{E}}$ on RHS of rule ($\mathcal{E}1$) to $\phi'_{\mathcal{E}} = \phi_{\mathcal{E}}[x \mapsto \{(\phi'_{\mathcal{S}}(x), U, U')\}]$. This further leads to the conclusion above. Similar line of reasoning can be followed in the cases of rules ($\mathcal{E}8$)–($\mathcal{E}15$) with respect to rules ($\mathcal{S}8$)–($\mathcal{S}15$). The case for replication in rules ($\mathcal{E}5$) and ($\mathcal{S}5$) requires the use of mathematical induction on the number of copies of P, with the basis being the case of zero copies and the inductive step proving that if the property holds for n copies of P, then it holds for $n + 1$ copies as well. Finally, in all other cases, neither $\phi_{\mathcal{S}}$ nor $\phi_{\mathcal{E}}$ change, so the conclusion is reached in a straightforward manner with the use of the antecedent. □

7 Abstract Semantics

One problem with the non-standard semantics of the previous section is that the calculation of the meaning of a protocol is not guaranteed to terminate due to the presence of replication in the definition of processes. Therefore, it is necessary to introduce a safe abstraction that limits the size of the semantic domain. This abstraction is a variation of the abstraction used in [3, 7].

In order to arrive a simple abstraction for terms later on, we begin first by assuming a predomain of tags, *Tag*, ranged over by $t, \dot{t}, \ddot{t}$, where t is the tag of a generic term, $\dot{t}$ is the tag of a name or a variable, and $\ddot{t}$ is the tag of a complex term (ciphertext, signature, pair). Next, we tag each M, M' in the processes $\overline{N}\langle M\rangle.P$, *let* $(x, y) = (M, M')$ *in* P, *case* $\{M\}_N$ *of* $\{x\}_{N'}$ *in* P, *case* $\{\![M]\!\}_N$ *of* $\{\![x]\!\}_{N'}$ *in* P, *case* $[\![\{M\}]\!]_N$ *of* $[\![\{x\}]\!]_{N'}$ *in* P, $A(M)$, and tag each of **private**(M), **public**(M) and **certified**(M) in the syntax.

The following functions are defined over tags and systems:
– $value\_of(\{t_1, \ldots, t_n\}) = \{M_1, \ldots, M_n\}$, which when applied to a set of tags, $\{t_1, \ldots, t_n\}$, returns the corresponding set of syntactic terms, $\{M_1, \ldots, M_n\}$.
– $tags\_of(E) = \{t_1, \ldots, t_n\}$, which when applied to a system, E, returns its set of tags, $\{t_1, \ldots, t_n\}$.

We now introduce the $\alpha_{k,k'}$ abstraction function, which keeps to a finite level, the number of copies of bound variables, bound names and tags, which will be indexed when interpreting replication.

Definition 2 *Define* $\alpha_{k,k'} : \mathbb{N} \times \mathbb{N} \times (V + N + Tag) \rightarrow (V^\sharp + N^\sharp + Tag^\sharp)$*:*

$$\forall M \in (V+N+Tag), i, k, k' \in \mathbb{N} : \alpha_{k,k'}(M) = \begin{cases} \dot{t}_k, & \text{if } M = \dot{t}_i \in Tag \text{ and } i > k \\ \ddot{t}_{k'}, & \text{if } M = \ddot{t}_i \in Tag \text{ and } i > k' \\ x_k, & \text{if } M = x_i \in V \text{ and } i > k \\ a_k, & \text{if } M = a_i \in N \text{ and } i > k \\ M, & \text{otherwise} \end{cases}$$

The resulting abstract predomains, $V^\sharp$, $N^\sharp$ and $Tag^\sharp$, can be defined as $V^\sharp = V\backslash\{x_j \mid j > k\}$, $N^\sharp = N\backslash\{a_j \mid j > k\}$ and $Tag^\sharp = Tag\backslash(\{\dot{t}_j \mid j > k\} \cup \{\ddot{t}_i \mid i > k'\})$. Informally, k constrains the number of bound variables and names, and tags of primitive terms, whereas k' constrains the number of tags of complex terms. In effect, constraining the tags of primitive terms implies limiting the copies of bound names and variables carrying the tags, whereas constraining the number of tags of complex terms means limiting the depth of data structures.

For example, in the process $!(\nu n)\overline{a}\langle n^{\dot{t}}\rangle \mid !a(x)$, it is possible to spawn infinite copies of each replication, $(\nu n_1)\overline{a}\langle n_1^{\dot{t}_1}\rangle \mid a(x_1) \mid (\nu n_2)\overline{a}\langle n_2^{\dot{t}_2}\rangle \mid a(x_2) \mid \ldots$. It is clear that the number labelling on $\dot{t}$ is an indicator to the number of the copy of message n after each process has been spawned. On the other hand, the process $!a(x).\overline{a}\langle\{x\}_k^{\ddot{t}}\rangle \mid \overline{a}\langle b\rangle$, which also spawns the copies $a(x_1).\overline{a}\langle\{x_1\}_k^{\ddot{t}_1}\rangle \mid a(x_2).\overline{a}\langle\{x_2\}_k^{\ddot{t}_2}\rangle \mid \overline{a}\langle b\rangle \mid \ldots$, demonstrates the role of $\ddot{t}$ as an indicator to the number of times the ciphertext, $\{x\}_k$, is applied to b.

It is essential to note at this stage that the usage of $\alpha_{k,k'}$ will inevitably reduce the precision of the semantics as a result of introducing approximate

behaviour. For example, the abstract meaning of the protocol based on the abstraction function, $\alpha_{1,1}$:

$$(\theta, \lceil \overline{c_1}\langle k\rangle.P\rceil^U \mid \lceil c_2(x).Q\rceil^{U'})$$

will include the false information that k substitutes x, since the calculation of the meaning will be based on abstracting both c_1 and c_2 to c_1. Therefore, a communication between either sides of the parallel composition will occur in the abs tart semantics, even though such a communication is not possible in the concrete (i.e. standard and non-standard) semantics.

Using $\alpha_{k,k'}$, we construct $\phi_{\mathcal{A}} : V^\sharp \to \wp(\mathit{Tag}^\sharp \times AG \times AG)$, with a meaning similar to $\phi_{\mathcal{E}}$ in the previous section. Furthermore, a domain, $D^\sharp_\perp = V^\sharp \to \wp(\mathit{Tag}^\sharp \times AG \times AG)$ is formed as follows:

$$\forall \phi_{\mathcal{A}1}, \phi_{\mathcal{A}2} \in D^\sharp_\perp, x \in V^\sharp : \phi_{\mathcal{A}1} \sqsubseteq_{D^\sharp_\perp} \phi_{\mathcal{A}2} \Leftrightarrow \phi_{\mathcal{A}1}(x) \subseteq \phi_{\mathcal{A}2}(x)$$

with a bottom element, $\perp_{D^\sharp_\perp}$, representing the null environment, $\phi_{\mathcal{A}0}$. Taking $D^\sharp_\perp$ as the abstract semantic domain, we can define the abstract semantics of the SPIKY language using the function, $\mathcal{A}[\![E]\!]\ \rho\ \phi_{\mathcal{A}}\ \theta \in D^\sharp_\perp$, as shown in Figure 11. The definitions of ρ and θ are as in the previous sections. The special function, $\varphi_{\mathcal{A}}$, returns a set of terms corresponding to a term, M, given substitutions captured by $\phi_{\mathcal{A}}$, as follows, where $\mathit{fst}(a,b,c) = a$:

$\varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) = \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M[\alpha_{k,k'}(t)/t][\alpha_{k,k'}(x)/x][\alpha_{k,k'}(n)/n])_{\{\}}$,
where, $\varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, M)_s = \mathit{if}\ M \in s\ \mathit{then}\ \{\}\ \mathit{else}$

$$\begin{cases} \bigcup_{L \in value\_of(fst(\phi_{\mathcal{A}}(M)))} \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}} & \text{if } M \in \mathcal{V} \\ \{M\}, & \text{if } M \in (\mathcal{N} \cup \mathcal{AG}) \\ \{\{N'\}^t_{L'} \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if } M = \{N\}^t_L \\ \{\{\![N']\!\}^t_{L'} \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if } M = \{\![N]\!\}^t_L \\ \{[\![N']\!]^t_{L'} \mid N' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, N)_{s\cup\{M\}}, L' \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L)_{s\cup\{M\}}\}, & \text{if } M = [\![N]\!]^t_L \\ \{(L'_1, L'_2)^t \mid L'_1 \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L_1)_{s\cup\{M\}}, L'_2 \in \varphi'_{\mathcal{A}}(\phi_{\mathcal{A}}, L_2)_{s\cup\{M\}}\}, & \text{if } M = (L_1, L_2)^t \end{cases}$$

We describe a few rules here. Rule ($\mathcal{A}1$) deals with the case of output actions, dealing with possible communications with appropriate input actions in ρ. The tag of the output message is registered in $\phi_{\mathcal{A}}$ as a value for the input variable. The semantics is imprecise, since $\phi_{\mathcal{A}}$ only captures an abstract tag as a value for an abstract variable. Rule ($\mathcal{A}5$) introduces the functions:

$\mathit{ren}(x,i) = \mathit{fold}\ \mathit{sub}_i\ (\mathit{fold}\ \mathit{sub}_i\ x\ \mathbf{bnv}(x))\ \mathit{tags\_of}(x)$
$\mathit{fold}\ f\ e\ \{x_1, \ldots, x_n\} = f(x_n, \ldots, f(x_1, e)\ldots)$
$\mathit{sub}_i\ x\ y = y[x_i/x]$

that are used in the definition of the least fixed point meaning of a replicated process. This meaning is defined as the least upper bound of the set $\mathcal{F}$, which can only be finite in this semantic. As a result, the termination of the least fixed point is formalised as follows.

$$
\begin{array}{lll}
(\mathcal{A}0A) & \mathcal{A}(\!|\lceil E \mid F\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \mathcal{R}(\!|\{\!|\lceil E\rceil^U|\!\} \uplus \{\!|\lceil F\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}\ \theta \\
(\mathcal{A}0B) & \mathcal{A}(\!|(\nu n)E|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \mathcal{R}(\!|\{\!|\lceil E\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}\ \theta \\
(\mathcal{A}1) & \mathcal{A}(\!|\lceil \overline{M}\langle L^t\rangle.P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \bigcup_{\phi \atop \lceil M'(z).P'\rceil^{U'} \in \rho : \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) \cap \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M') \cap \mathcal{N} \neq \{\}} \phi'_{\mathcal{A}}\ \cup_{\phi}\ \phi_{\mathcal{A}} \\
 & \text{where, } \phi'_{\mathcal{A}} = \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho[\lceil P'\rceil^{U'}/\lceil M'(z).P'\rceil^{U'}]|\!)\ \phi''_{\mathcal{A}}\ \theta & \\
 & \text{and } \phi''_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(z) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(z)) \cup \{(\alpha_{k,k'}(t), U, U')\}] & \\
(\mathcal{A}2) & \mathcal{A}(\!|\lceil M(y).P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \phi_{\mathcal{A}} \\
(\mathcal{A}3) & \mathcal{A}(\!|\lceil P \mid Q\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \{\!|\lceil Q\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}\ \theta \\
(\mathcal{A}4) & \mathcal{A}(\!|\lceil (\nu n)P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}\ \theta \\
(\mathcal{A}5) & \mathcal{A}(\!|\lceil !P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \bigsqcup \mathcal{F} \\
 & \text{where, } \mathcal{F} = \{\bot_{D^\sharp_\bot}, \mathcal{A}(\!|\prod_i ren(P,i)\ |\!)\ \rho\ \phi_{\mathcal{A}}\ \theta \mid\ i = 0 \ldots \infty\} & \\
(\mathcal{A}6) & \mathcal{A}(\!|\lceil [M\ is\ N]P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = \begin{cases} \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}\ \theta, \\ \text{if } \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) \cap \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \cap \mathcal{N} \neq \{\} \\ \phi_{\mathcal{A}}, \text{ otherwise} \end{cases} & \\
(\mathcal{A}7) & \mathcal{A}(\!|\lceil \mathbf{0}\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta & = \phi_{\mathcal{A}} \\
(\mathcal{A}8) & \mathcal{A}(\!|\lceil let\ (x,y) = M\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \begin{cases} \bigcup_{\phi \atop (L^t, N^{t'}) \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M)} \mathcal{R}(\!|\{\!|P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta, & \text{if } \exists (L^t, N^{t'}) \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{\alpha_{k,k'}(t), U, U\}, & \\ \alpha_{k,k'}(y) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(y)) \cup \{\alpha_{k,k'}(t'), U, U\}] & \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} & \\
(\mathcal{A}9) & \mathcal{A}(\!|\lceil case\ L\ of\ \{x\}_N\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \begin{cases} \bigcup_{\phi \atop \{M^t\}_n \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)} \mathcal{R}(\!|\{\!|P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta, & \text{if, } n \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}] & \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} & \\
(\mathcal{A}10) & \mathcal{A}(\!|\lceil case\ L\ of\ \{\!|x|\!\}_N\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \begin{cases} \bigcup_{\phi \atop \{\!|M^t|\!\}_{n^+} \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)} \mathcal{R}(\!|\{\!|P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta, & \text{if, } n^- \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}] & \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} & \\
(\mathcal{A}11) & \mathcal{A}(\!|\lceil case\ L\ of\ \{\!|x|\!\}_N\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \begin{cases} \bigcup_{\phi \atop \{\!|M^t|\!\}_{n^-} \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, L)} \mathcal{R}(\!|\{\!|P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta, & \text{if, } n^+ \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, N) \\ \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}] & \\ \phi_{\mathcal{A}}, & \text{otherwise} \end{cases} & \\
(\mathcal{A}12) & \mathcal{A}(\!|\lceil A(M^t)\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta \text{ where, } A(x) \triangleq P & \\
 & \text{and } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}] & \\
(\mathcal{A}13) & \mathcal{A}(\!|\lceil let\ x = \mathbf{private}(M)^t\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \begin{cases} \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}]\ \theta, & \\ \quad \text{if } U \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M), \text{ where, } \mathbf{private}(U) = \theta(U)^- & \\ \phi_{\mathcal{A}}, \text{ otherwise} & \end{cases} & \\
(\mathcal{A}14) & \mathcal{A}(\!|\lceil let\ x = \mathbf{public}(M)^t\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi'_{\mathcal{A}}\ \theta & \\
 & \text{where, } \phi'_{\mathcal{A}} = \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}] & \\
 & \text{and, } value\_of(\{t\}) = \begin{cases} \{\theta(U)^+\}, \text{ if } U \in \varphi_{\mathcal{A}}(\phi_{\mathcal{A}}, M) \\ \{\theta(U')^+ \mid U' \in dom(\theta)\}, \text{ otherwise} \end{cases} & \\
(\mathcal{A}15) & \mathcal{A}(\!|\lceil let\ x = \mathbf{certified}(M)^t\ in\ P\rceil^U|\!)\ \rho\ \phi_{\mathcal{A}}\ \theta = & \\
 & \mathcal{R}(\!|\{\!|\lceil P\rceil^U|\!\} \uplus \rho|\!)\ \phi_{\mathcal{A}}[\alpha_{k,k'}(x) \mapsto \phi_{\mathcal{A}}(\alpha_{k,k'}(x)) \cup \{(\alpha_{k,k'}(t), U, U)\}]\ \theta, & \\
 & \text{where, } \mathbf{certified}(M) = \theta(M)^+ & \\
(\mathcal{R}0) & \mathcal{R}(\!|\rho|\!)\ \phi_{\mathcal{A}}\ \theta & = \bigcup_{\phi \atop \lceil P\rceil^U \in \rho} \mathcal{A}(\!|\lceil P\rceil^U|\!)\ (\rho \backslash \{\!|\lceil P\rceil^U|\!\})\ \phi_{\mathcal{A}}\ \theta
\end{array}
$$

Figure 11: The abstract semantics of the SPIKY language.

Theorem 2 (Termination of the least fixed point calculation)
The calculation of rule ($\mathcal{A}5$) *terminates.*

Proof. We provide here a sketch of the termination property. To prove this, it is necessary to show that the following two requirements hold. First, that the semantic domain is finite. This is satisfied by the definition of $D^\sharp_\perp$, where $Tag^\sharp$ and AG are both finite, and so is $\wp(Tag^\sharp \times AG \times AG)$. The second requirement is that $\mathcal{A}(\![\lceil \prod_i P \rceil^U]\!)\ \rho\ \phi_\mathcal{A}\ \theta$ is monotonic over P, i.e. $\mathcal{A}(\![\lceil \prod_i P \rceil^U]\!)\ \rho\ \phi_\mathcal{A}\ \theta \sqsubseteq_{D^\sharp_\perp} \mathcal{A}(\![\lceil \prod_{i+1} P \rceil^U]\!)\ \rho\ \phi_\mathcal{A}\ \theta$. To prove the second requirement, we simply the ordering relation into $\mathcal{A}(\![E]\!)\ \rho\ \phi_\mathcal{A}\ \theta \sqsubseteq_{D^\sharp_\perp} \mathcal{A}(\![E \mid \lceil P \rceil^U]\!)\ \rho\ \phi_\mathcal{A}\ \theta$, where $E = \lceil \prod_i P \rceil^U$. This is further simplified to become $\mathcal{A}(\![E]\!)\ \rho\ \phi_\mathcal{A}\ \theta \sqsubseteq_{D^\sharp_\perp} \mathcal{A}(\![E]\!)\ \rho'\ \phi_\mathcal{A}\ \theta$, where $\rho' = \rho \uplus \lceil P \rceil^U$. This can be proven by considering every case of E in particular, the case of rule ($\mathcal{A}1$), which deals with output action is an interesting case, since it is possible to compare the communications with systems in ρ and in ρ'. When comparing these two multisets, we find that $\phi_\mathcal{A}$ for the former is a subset of the latter, i.e. the larger the ρ is, the more communications we have. □

We can state the safety of the abstract semantics by the following theorem.

Theorem 3 (Safety of the abstract semantics)
$\forall P, \theta, \rho, \phi_\mathcal{E}, \phi_\mathcal{A}, k, k', U, \phi'_\mathcal{E} = \mathcal{E}(\![\lceil P \rceil^U]\!)\ \rho\ \phi_\mathcal{E}\ \theta, \phi'_\mathcal{A} = \mathcal{A}(\![\lceil P \rceil^U]\!)\ \rho\ \phi_\mathcal{A}\ \theta :$
$(\exists M, x, U, U' : (\varphi_\mathcal{E}(\phi_\mathcal{E}, M), U, U') \in \phi_\mathcal{E}(x)\ \Rightarrow$
$\exists (t, U, U') \in \phi_\mathcal{A}(\alpha_{k,k'}(x)) : M^\sharp \in \mathit{value\_of}(\{t\}) \wedge M^\sharp = \mathit{fold}\ \mathit{sub}_{k,k'}\ M\ \boldsymbol{nv}(M))$
$\Rightarrow$
$(\exists M, x, U, U' : (\varphi_\mathcal{E}(\phi'_\mathcal{E}, M), U, U') \in \phi'_\mathcal{E}(x)\ \Rightarrow$
$\exists (t, U, U') \in \phi'_\mathcal{A}(\alpha_{k,k'}(x)) : M^\sharp \in \mathit{value\_of}(\{t\}) \wedge M^\sharp = \mathit{fold}\ \mathit{sub}_{k,k'}\ M\ \boldsymbol{nv}(M))$
where, $\mathit{sub}_{k,k'}\ x\ y = y[\alpha_{k,k'}(x)/x]$
and, $nv(M)$ *is the set of names and variables of* M

Proof. The proof is by the induction on the structure of systems. We provide a sketch of the proof as follows. The base case is that for $E = \lceil \mathbf{0} \rceil$, where the values of $\phi_\mathcal{A}$ and $\phi_\mathcal{E}$ are left intact in rules ($\mathcal{A}7$) and ($\mathcal{E}7$) moving from LHS to RHS of each rule. The antecedent is needed in this case to reach to the conclusion. The inductive step considers every other case of a system. The most interesting cases are those where the values of $\phi_\mathcal{A}$ and $\phi_\mathcal{E}$ are changed. These include the cases of rules, ($\mathcal{A}1$), ($\mathcal{A}8$)–($\mathcal{A}15$) and their concrete parts, ($\mathcal{E}1$), ($\mathcal{E}8$)–($\mathcal{E}15$), respectively. In each of these cases, it is possible to show that if $\phi_\mathcal{A}$ satisfies the antecedent requirement initially with respect to some $\phi_\mathcal{E}$, then the new $\phi'_\mathcal{A}$ will satisfy the conclusion with respect to the new $\phi'_\mathcal{E}$, where $\phi'_\mathcal{A}$ and $\phi'_\mathcal{E}$ contain the changes dictated by the rule. In the case of every other rule where neither $\phi_\mathcal{A}$ nor $\phi_\mathcal{E}$ change, then the antecedent guarantees that the conclusion of the safety requirement is reachable. The most difficult case is that of replication, where additional mathematical induction reasoning is applied over the number of copies of the replicated system. □

The theorem states that for any term, M, captured in the non-standard semantics by including its $\varphi_\mathcal{E}(\phi'_\mathcal{E}, M)$ value in the value of a variable, $\phi'_\mathcal{E}(x)$,

then that corresponds to capturing a tag, t, in the abstract semantics, by $\phi'_{\mathcal{A}}(\alpha_{k,k'}(x))$. The appropriateness of t is expressed by the ability to obtain (by folding) an abstract form, $M^\sharp = \mathit{fold}\ \mathit{sub}_{k,k'}\ M\ \mathbf{nv}(M)$, of the concrete term, M, by evaluating t using $\mathit{value\_of}$. More concisely, every concrete term, M, captured in the non-standard semantics is also captured in the form of the corresponding abstract tag, t, in the abstract semantics. From now on, we shall use the following predicate, to denote the property of term capturing.

Definition 3 (Term capturing) *A term, M, is captured by an agent, B, sent by another agent, A, if given the results of an abstract interpretation, $\phi_{\mathcal{A}}$, and the integer constraints, k and k', then the following holds true:*

$$\mathit{captured}^\sharp(M, A, B, \phi_{\mathcal{A}}, k, k') \stackrel{\mathit{def}}{=}$$
$$\exists t \in \mathit{Tag}^\sharp, x \in \mathit{dom}(\phi_{\mathcal{A}}) : (t, A, B) \in \phi_{\mathcal{A}}(x) \ \wedge\ M^\sharp \in \mathit{value\_of}(t)$$

We also provide a more precise version of $\mathit{captured}^\sharp$, which we term $\mathit{only\_captured}^\sharp$ and that signifies the fact that a variable is instantiated with exactly one term.

Definition 4 (Precise term capturing) *A term, M, is "only" captured by an agent, B, sent by another agent, A, if given the results of an abstract interpretation, $\phi_{\mathcal{A}}$, and the integer constraints, k and k', then the following holds true:*

$$\mathit{onlycaptured}^\sharp(M, A, B, \phi_{\mathcal{A}}, k, k') \stackrel{\mathit{def}}{=}$$
$$\exists t \in \mathit{Tag}^\sharp, x \in \mathit{dom}(\phi_{\mathcal{A}}) : (t, A, B) \in \phi_{\mathcal{A}}(x) \ \wedge\ |\phi_{\mathcal{A}}(x)| = 1 \ \wedge\ M^\sharp \in \mathit{value\_of}(t)$$

Where the extra condition that $|\phi_{\mathcal{A}}(x)| = 1$ implies that x can only ever be replaced by a single term (M in this case).

8 The Intruder

There are often two approaches to the modelling of intruders in any security analysis: the first approach aims at encoding the behaviour of the intruder into the semantic rules describing the analysis itself, and the second models the intruder as any other process. Each approach has its advantages. The first approach results in an analysis that is specialised to deal with the intruder without the need to explicitly model the intruder. However, this approach is rigid. The second approach offers more flexibility in considering any intruder from the simplest passive intruders to Dolev-Yao's most powerful intruder [14].

In this report, we adopt the second approach in modelling the intruder within our analysis. The model describes the general guidelines along which the most general attacker in cryptographic protocols can be specified. This model was shown by [13] to be sufficient to subsume any other adversary. Informally, a specification of the Dolev-Yao attacker should adhere to the following criteria:

- The attacker can read, learn, modify and block any messages passed over the network's public channels, as well as create fresh messages. It can also send the messages it has in its knowledge to other processes.

- The attacker can compose tuples from learnt messages and can decompose learnt tuples to their basic elements.
- The attacker can apply cryptographic operations to any of the messages it has in its knowledge using any of the keys it knows about.
- The intruder will always attempt to retrieve the public/private keys of any of the agent names it knows about from the underlying PKI.

The above features can be stated more formally in the SPIKY language by the specification of the intruder's system, $ISYS$, as illustrated in Figure 12, where $\prod$ denotes the parallel composition of several processes.

$$
\begin{array}{l}
ISYS \stackrel{\texttt{def}}{=} \lceil (\nu\, i)\ (\ \bar{i}\langle \kappa_{init}\rangle \ \mid\ !i(\kappa).(\ (\nu\, net)\bar{i}\langle \kappa, net\rangle \ \mid \\
\quad \prod_{\forall M,N \in set(\kappa)} \overline{M}\langle N\rangle.\bar{i}\langle \kappa\rangle \ \mid\ \prod_{\forall M \in set(\kappa)} M(x).\bar{i}\langle \kappa, x\rangle \ \mid \\
\quad \prod_{\forall M,N,L \in set(\kappa)} \overline{M}\langle \{N\}_L\rangle.\bar{i}\langle \kappa, \{N\}_L\rangle \ \mid\ \prod_{\forall M,N,L \in set(\kappa)} \overline{M}\langle \{\!|N|\!\}_L\rangle.\bar{i}\langle \kappa, \{\!|N|\!\}_L\rangle \ \mid \\
\quad \prod_{\forall M,N,L \in set(\kappa)} \overline{M}\langle [\![N]\!]_L\rangle.\bar{i}\langle \kappa, [\![N]\!]_L\rangle \ \mid\ \prod_{\forall M,N,L \in set(\kappa)} \overline{M}\langle (N, L)\rangle.\bar{i}\langle \kappa, (N, L)\rangle \ \mid \\
\quad \prod_{\forall M,N \in set(\kappa)} case\ M\ of\ \{x\}_N\ in\ \bar{i}\langle \kappa, x\rangle \ \mid\ \prod_{\forall M,N \in set(\kappa)} case\ M\ of\ \{\!|x|\!\}_N\ in\ \bar{i}\langle \kappa, x\rangle \ \mid \\
\quad \prod_{\forall M,N \in set(\kappa)} case\ M\ of\ [\![x]\!]_N\ in\ \bar{i}\langle \kappa, x\rangle \ \mid\ \prod_{\forall M \in set(\kappa)} let\ (x, y) = M\ in\ \bar{i}\langle \kappa, (x, y)\rangle \ \mid \\
\quad \prod_{\forall M \in set(\kappa)} let\ x = \mathbf{private}(M)\ in\ \bar{i}\langle \kappa, x\rangle \ \mid\ \prod_{\forall M \in set(\kappa)} let\ x = \mathbf{public}(M)\ in\ \bar{i}\langle \kappa, x\rangle \ \mid \\
\quad \prod_{\forall M \in set(\kappa)} let\ x = \mathbf{certified}(M)\ in\ \bar{i}\langle \kappa, x\rangle)) \rceil^{I}
\end{array}
$$

Figure 12: Specification of the system of the Dolev-Yao attacker in SPIKY.

In this specification, I is the name of the intruder agent and κ_{init} is the initial knowledge of the intruder represented as a pair, $((M1, M_2), \ldots), M_n)$. If $n = 0$, then we write $\kappa_{init} = (,)$. Usually, κ_{init} is initialised with elements from the set of free names of the system, E, running in parallel with the intruder, i.e. $\kappa_{init} = ((M_1, M_2), \ldots), M_n)$, where $\mathbf{fn}(E) = \{M_1, M_2, \ldots, M_n\}$[3]. The specification also contains the subprocess, $\bar{i}\langle \kappa_{init}\rangle$, which initialises the knowledge of the intruder by communicating with the input process, $i(\kappa)$, and hence, yielding the substitution, $\kappa = \kappa_{init}$.

Moreover, we refer to the set of terms underlying κ (resp. κ_{init}) as $set(\kappa)$ (resp. $set(\kappa_{init})$). The knowledge of the intruder, κ, is increased due to the value-passing behaviour whenever input actions occur or fresh data are created as part of bound output actions. κ also increases due to the value-processing behaviour whenever decryption, signature verification, pair-splitting or any of the PKI-based operations succeed. In any case, standard pair concatenation, $(\_, \_) : Term \times Term \rightarrow Term$, is used to model the knowledge increase.

[3]Note that the order of the pair elements in κ_{init} is not important since κ_{init} is used to merely simulate a set.

Apart from the initialisation process $\bar{i}\langle\kappa_{init}\rangle$, the rest of the specification consists of a replication of processes each of which is guarded by an input action, $i(\kappa)$, over the special channel i. The input parameter κ is updated with a pair of terms, which is a necessary behaviour in order to be able to express the fact that I can learn from its own behaviour. For example, in order for κ to obtain the new name, *net*, without necessarily outputting *net* to external processes, I sends *net* over channel i. Similarly, in order for κ to learn all the terms it has encrypted, signed etc., it needs to send them again over channel i. On the other hand, the main body of the process consists of the parallel composition of all the possible input/output actions and cryptographic/PKI operations quantified over all the terms in κ.

An important point to note at this stage is that the specification of Figure 12 is not unique. It is possible to adopt other specifications of the intruder. Among these, $ISYS \stackrel{\texttt{def}}{=} \lceil \mathbf{0} \rceil^I$ is the weakest intruder, which is incapable of performing any action and,

$$ISYS \stackrel{\texttt{def}}{=} \lceil (\bar{i}\langle\kappa_{init}\rangle \mid !i(\kappa).(\prod_{\forall M,N \in set(\kappa)} \overline{M}\langle N\rangle.\bar{i}\langle\kappa\rangle \mid \prod_{\forall M \in set(\kappa)} M(x).\bar{i}\langle\kappa, x\rangle)) \rceil^I$$

defines the passive intruder, which can only input and output messages without performing any actions on them.

9 Abstract Security Properties

In this section, we define the security properties of abstract term secrecy and abstract peer-entity participation in light of the results of our abstract semantics.

9.1 Abstract Term Secrecy

Term secrecy refers to the property that a particular term is never leaked to some agent during the execution of a protocol. Using the *captured*$^\sharp$ predicate defined in Definition (3), we formalise abstract term secrecy of a term, M, with respect to an agent, U, written as *secret*$^\sharp(M, U)$, as follows.

Definition 5 (Abstract term secrecy)
We say that a term, M, remains abstractly secret with respect to an agent, U, written as secret$^\sharp(M, U)$, if the following holds true:

$$\nexists U' \in AG : \ captured^\sharp(M, U', U, \phi_{\mathcal{A}}, k, k')$$

9.2 Abstract Peer-Entity Participation

Peer-entity participation means that an agent, A, knows to a certain degree of certainty that another agent, B, has participated in a session of some protocol in which A is also a participant. In reality, there are many scenarios that this property could be established, both in its one-way and two-way forms. In this

section, we discuss one such scenario, where A creates a nonce, n, and n is signed by B, then, provided that only B has the knowledge of its own private key, A knows that B has just participated in the protocol if it receives back a signed term, M, containing n and is able to verify it with B's public key. We define here three cases of the abstract peer-entity participation property.

Definition 6 (Abstract peer-entity participation)
Given the abstract interpretation, $\phi_{\mathcal{A}} = \mathcal{A}[\![C(\lceil(\nu n)P\rceil^A)]\!]\ \rho_0\ \phi_{\mathcal{A}0}\ \theta$, for some context, C, and process, P, then we say that the agent, A, is said to be sure to a certain degree that another agent, B, has participated in the protocol, $(\theta, C((\nu n)\lceil P\rceil^A))$, written as participated$^\sharp(A,B)$, if the following holds true:

$$\exists U, U', L, k, k' : \ captured^\sharp(n, U, B, \phi_{\mathcal{A}}, k, k') \wedge captured^\sharp(\theta(B)^-, B, B, \phi_{\mathcal{A}}, k, k') \wedge captured^\sharp([\![\{L\}]\!]_{\theta(B)^-}, U', A, \phi_{\mathcal{A}}, k, k') \ \wedge \ captured^\sharp(L, A, A, \phi_{\mathcal{A}}, k, k') \ \wedge \ (\forall U'' : U'' \neq B \Rightarrow secret^\sharp(\theta(B)^-, U'')) \ \wedge \ n \in \mathbf{n}(L)$$

And either one of the following three requirements:

1- $captured^\sharp(\theta(B)^+, A, A, \phi_{\mathcal{A}}, k, k')$ *(Non-certified case)*

2- $onlycaptured^\sharp(\theta(B)^+, A, A, \phi_{\mathcal{A}}, k, k')$ *(Certified case)*

3- $\exists S :\ onlycaptured^\sharp(\theta(B)^+, S, S, \phi_{\mathcal{A}}, k, k')$ *(Delegated case) where S is a server trusted by A*

The definition of *participated*$^\sharp$ essentially states that B must obtain n as well as its own private key, signs a term, L, containing n with the private key and finally, A must obtain this term and correctly verifies it with the public key of B. This is of course based on the condition that B keeps its own private key secret. The degree at which A is certain of the participation of B depends on whether the public key of B was *captured* (i.e., using $\mathbf{public}(B)$ since $\mathbf{public}(B)$ may return an invalid key) or *only captured* (i.e. using $\mathbf{certified}(B)$ since $\mathbf{certified}(B)$ always returns a valid key) and whether A trusts in some third party, S. Three scenarios then arise: the certified case, the uncertified case and the delegated case.

10 Examples

We consider here again the two examples introduced earlier in Section 4, where we apply the abstract interpretation developed in Section 7 to these protocols and analyse their abstract term secrecy and peer-entity participation properties.

10.1 The Simple Authentication Protocol

We apply the abstract interpretation, $\mathcal{A}[\![SYST(A,B)]\!]\ \{\!|ISYS|\!\}\ \phi_{\mathcal{A}0}\ \theta$, to the specification of the simple authentication protocol, $(\theta, SYST(A,B))$, for some two agents, A and B, and for the uniform case where $k = k' = 1$. The final

fixed point value for $\phi_{\mathcal{A}}$ is given as follows, after we have applied the *value_of* function to retrieve the values of terms from their tags:

$$\phi_{\mathcal{A}} = \left[\begin{array}{ll}
yb \mapsto \{(B,B,B)\} & xb \mapsto (B,A,A) \quad xa \mapsto \{(A,A,A)\} \\
ych \mapsto \{(ch,B,B)\} & xch \mapsto \{(ch,A,A)\} \\
ya \mapsto \{(A,A,B)\} & xb' \mapsto \{(B,B,A)\} \\
yn_a \mapsto \{(n_a,A,B)\} & xn_b \mapsto \{(n_b,B,A)\} \\
yk_b \mapsto \{(\theta(B)^-,B,B)\} & xsig \mapsto \{([\![n_a]\!]_{\theta(B)^-},B,A)\} \\
ysig \mapsto \{([\![n_b]\!]_{\theta(A)^-},A,B)\} & xk_b \mapsto \{(\theta(B)^+,A,A)\} \\
yk_a \mapsto \{(\theta(A)^+,B,B)\} & x \mapsto \{(n_a,A,A)\} \\
y \mapsto \{(n_b,B,B)\} & xk_a \mapsto \{(\theta(A)^-,A,A)\} \\
\kappa \mapsto \{(I,I,I),(A,A,I),(N_A,A,I),(B,B,I),(N_B,B,I), & \\
\quad (\theta(A)^+,I,I),(\theta(B)^+,I,I),(\theta(I)^+,I,I),(\theta(I)^-,I,I)\} &
\end{array}\right]$$

According to Definition (5), the results confirm that $secret^\sharp(U,\theta(A)^-)$ and $secret^\sharp(U',\theta(B)^-)$ for any $U \in \{B,I\}$ and $U' \in \{A,I\}$. This implies that neither agent was able to obtain the other agent's private key nor that the intruder was able to obtain the secret keys. Furthermore, one can also see that $participated^\sharp(A,B)$ and $participated^\sharp(B,A)$ are true, since both fulfil case 2 of Definition (6).

10.2 The Mobile Authentication Protocol

For the case of the mobile authentication protocol, we apply the abstract interpretation, $\mathcal{A}([\![SYST(A,B,S)]\!])\ \{\!|ISYS|\!\}\ \phi_{\mathcal{A}0}\ \theta$, for the uniform case, i.e. where $k = k' = 1$. The fixed point results for $\phi_{\mathcal{A}}$ after the application of the *value_of* function are:

$\phi_{\mathcal{A}} =$

$$\left[\begin{array}{ll}
xa \mapsto \{(A,A,A)\} & xs \mapsto \{(S,A,A)\} \\
yb \mapsto \{(B,B,B)\} & xb \mapsto \{(B,A,A)\} \\
ych \mapsto \{(ch,B,B)\} & xch \mapsto \{(ch,A,A)\} \quad xch' \mapsto \{(ch',A,A)\} \\
zs \mapsto \{(S,S,S)\} & zch \mapsto \{(ch,S,S)\} \quad zch' \mapsto \{(ch',S,S)\} \\
ya \mapsto \{(A,A,B)\} & xb' \mapsto \{(B,B,A)\} \\
yn_a \mapsto \{(n_a,A,B)\} & xn_b \mapsto \{(n_b,B,A)\} \\
yk_b \mapsto \{(\theta(B)^-,B,B)\} & xsig \mapsto \{([\![n_a]\!]_{\theta(B)^-},B,A)\} \\
ysig \mapsto \{([\![n_b]\!]_{\theta(A)^-},A,B)\} & xk_b \mapsto \sum_{U \in dom(\theta)} \{(\theta(U)^+,A,A)\} \\
yk_a \mapsto \{(\theta(A)^+,B,B)\} & x \mapsto \{(n_a,A,A)\} \quad y \mapsto \{(n_b,B,B)\} \\
xk_a \mapsto \{(\theta(A)^-,A,A)\} & xk_s \mapsto \{(\theta(S)^+,A,A)\} \\
zk_s \mapsto \{(\theta(S)^-,S,S)\} & \\
zc \mapsto \sum_{U \in dom(\theta)} \{([\![([\![n_b]\!]_{\theta(A)^-},B,\theta(U)^+)]\!]_{\theta(S)^+},A,S)\} & \\
zp \mapsto \{(([\![n_b]\!]_{\theta(A)^-},B,\theta(U)^+),S,S)\} & zsig \mapsto \{([\![n_b]\!]_{\theta(A)^-},S,S)\} \\
zb \mapsto \{(B,S,S)\} & zkey \mapsto \sum_{U \in dom(\theta)} \{(\theta(U)^+,S,S)\} \\
zk_b \mapsto \{(\theta(B)^+,S,S)\} & \\
\kappa \mapsto \{(I,I,I),(A,A,I),(N_A,A,I),(B,B,I),(N_B,B,I),(S,I,I) & \\
\quad (\theta(A)^+,I,I),(\theta(B)^+,I,I),(\theta(S)^+,I,I),(\theta(I)^+,I,I),(\theta(I)^-,I,I)\} &
\end{array}\right]$$

From these results, it is possible to verify that the secrecy requirement on all the private keys of the protocol i.e., $\mathit{secret}^\sharp(U, \theta(A)^-)$, $\mathit{secret}^\sharp(U', \theta(B)^-)$ and $\mathit{secret}^\sharp(U'', \theta(S)^-)$, where $U \in \{B, S, I\}$, $U' \in \{A, S, I\}$ and $U'' \in \{A, B, I\}$. In the case of B, we find that $\mathit{participated}^\sharp(B, A)$ can be shown to hold according to case (2) of Definition (6), however, in the case of the verification of B's participation by A, i.e. $\mathit{participated}^\sharp(A, B)$, it is only possible to show that case (1) holds true in Definition (6) $\mathit{participated}^\sharp(A, B)$, but that case (2) is impossible to prove. However, we find that case (3) can be proven for the case of A, since A relies on S.

11 Conclusion

In this technical report, we reviewed the process algebraic language, SPIKY, which is used for the modelling of systems that use Public-Key Infrastructures (PKIs). SPIKY formalises the notion of *process ownership* by PKI agents as well as the operations of (un)certified public and private key retrievals, which constitute an essential element of any PKI-based protocol. We then developed a domain-theoretic model for protocols in SPIKY and used the model to construct a non-uniform static analysis for capturing the property of term-substitutions. These substitutions appear as a result of message-passing and the application of cryptographic/PKI operations within protocols. The results of the static analysis were used to formalise two security properties: term secrecy and peer-entity participation. Finally we used these formalisations to reason about the security properties of a couple of simple authentication protocols assumed to be running in parallel with Dolev-Yao's attacker.

For future work, we would like to apply the analysis to more complicated protocols, such as the Internet Key Exchange protocol (IETF RFC 2409) and the Transport Layer Security protocol (IETF RFC 2246). We would also like to investigate the quantitative aspects of PKI-based operations by introducing a theory of constraint semirings in a manner similar to [4], which deals with the cost of mobility and message-passing. Such analyses are of particular interest when comparing the cost of security against the quantitative capabilities of the different models of intruders, e.g. the weakest, passive and the most powerful intruder.

References

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

[2] Samson Abramsky. A domain equation for bisimulation. *Information and Computation*, 92(2):161–218, June 1991.

[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] Benjamin Aziz. A semiring-based quantitative analysis of mobile systems. In *Proceedings of the* 3rd *International Workshop on Software Verification and Validation*, volume 157(1) of *Electronic Notes in Theoretical Computer Science*, pages 3–21, Manchester, UK, October 2005. Elsevier.

[5] Benjamin Aziz, David Gray, and Geoff Hamilton. A static analysis of pki-based systems. In *Proceedings of the* 9th *Italian Conference on Theoretical Computer Science*, volume 3701 of *Lecture Notes in Computer Science*, pages 51–65, Siena, Italy, October 2005. Springer Verlag.

[6] Benjamin Aziz and Geoff Hamilton. The modelling and analysis of pki-based systems using process calculi. *International Journal of Foundations of Computer Science*, 2007 (to appear).

[7] Benjamin Aziz, Geoff Hamilton, and David Gray. A static analysis of cryptographic processes: The denotational approach. *Journal of Logic and Algebraic Programming*, 64(2):285–320, August 2005.

[8] Hanane El Bakkali and Bahia Idrissi Kaitouni. A predicate calculus logic for the pki trust model analysis. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, pages 368–371, Cambridge, MA, USA, October 2001. IEEE Computer Society.

[9] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the* 16th *IEEE Computer Security Foundations Workshop*, pages 126–140, Pacific Grove, CA, USA, June 2003. IEEE Computer Society.

[10] 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.

[11] Chiara Bodei, Pierpaolo Dagano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 168(1):68–92, July 2001.

[12] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Jzef Winkowski, editors, *Proceedings of the* 8th *International Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150, Warsaw, Poland, July 1997. Springer Verlag.

[13] Iliano Cervesato. The dolev-yao intruder is the most powerful attacker. In J. Halpern, editor, *Proceedings of the* 16th *Annual Symposium on Logic in Computer Science*, pages 246–265, Boston, MA, U.S.A., June 2001. IEEE Computer Society Press.

[14] Danny Dolev and A. Yao. On the security of public key protocols. In *Proceedings of the* 22nd *Annual Symposium on Foundations of Computer Science*, pages 350–357, October 1981.

[15] C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomas, and T. Ylonen. Spki certificate theory. RFC 2693, September 1999.

[16] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the* 12th *International Conference on Computer-Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247, Chicago, IL, USA, July 2000. Springer Verlag.

[17] 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.

[18] M.J.C. Gordon and T.F. Melham. *Introduction to HOL.* Cambridge University Press, Cambridge, UK, 1993.

[19] David Gray, Benjamin Aziz, and Geoff Hamilton. Spiky: A nominal calculus for modelling protocols that use pkis. In *Proceedings of the International Workshop on Security Analysis of Systems: Formalism and Tools*, Orléans, France, June 2004.

[20] Steffen M. Hansen, Jakob Skriver, and Hanne Riis Nielson. Using static analysis to validate the saml single sign-on protocol. In *Proceedings of the 2005 workshop on Issues in the theory of security*, pages 27–40, Long Beach, California, USA, 2005. ACM Press.

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

[22] Somesh Jha and Thomas W. Reps. Analysis of spki/sdsi certificates using model checking. In *Proceedings of the* 15th *IEEE Computer Security Foundations Workshop*, pages 129–147, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society.

[23] T. Ksiyatrakul, S. Older, and S.-K. Chin. Formal analysis of x.509 certificates. Technical Report TR DII 08/01, Universit degli studi di Siena, 2001.

[24] Eve Maler, Prateek Mishra, and Rob Philpott. Assertions and protocol for the oasis security assertion markup language, version 1.1 edition. Technical Report oasis-sstc-saml-core-1.1, OASIS, September 2003.

[25] Eve Maler, Prateek Mishra, and Rob Philpott. Bindings and proles for the oasis security assertion markup language, version 1.1 edition. Technical Report sstc-saml-bindings-1.1-cs-01, OASIS, May 2003.

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

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

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

[29] Flemming Nielson and Hanne Riis Nielson. Flow logic and operational semantics. In Andrew Pitts Andrew Gordon and Carolyn Talcott, editors, *Electronic Notes in Theoretical Computer Science*, volume 10. Elsevier Science Publishers, 2000.

[30] Flemming Nielson, Hanne Riis Nielson, and René Rydhof Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(1):381–418, June 2002.

[31] Gordon Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, September 1976.

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

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

[34] Ian Stark. A fully abstract domain model for the π-calculus. In *Proceedings of the* 11^{th} *Annual IEEE Symposium on Logic in Computer Science*, pages 36–42, New Brunswick, New Jersey, USA, July 1996. IEEE Computer Society.

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

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