On Evaluating the Performance of Security Protocols *

Chiara Bodei[1], Mikael Buchholtz[3], Michele Curti[1], Pierpaolo Degano[1], Flemming Nielson[3], Hanne Riis Nielson[3], Corrado Priami[2]

1 Dipartimento di Informatica, Università di Pisa
Largo B.Pontecorvo, 3, I-56127 Pisa, Italy. – {chiara,curtim,degano}@di.unipi.it

2 Dipartimento di Informatica e Telecomunicazioni, Università di Trento
Via Sommarive, I-1438050 Povo (TN), Italy. – priami@science.unitn.it

3 Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads bldg 321, DK-2800 Kongens Lyngby, Denmark
– {mib,nielson,riis}@imm.dtu.dk

Abstract. We use an enhanced operational semantics to infer quantitative measures on systems describing cryptographic protocols. System transitions carry enhanced labels. We assign rates to transitions by only looking at these labels. The rates reflect the distributed architecture running applications and the use of possibly different crypto-systems. We then map transition systems to Markov chains and evaluate performance of systems, using standard tools.

1 Introduction

Cryptographic protocols are used in distributed systems for authentication and key exchange, and must therefore guarantee security. The mechanisms used are always the result of a judicious balance between their cost and benefits. Performance costs, in terms of time overhead and resource consumption, must be carefully evaluated when choosing security mechanisms.

Here, we extend a preliminary idea introduced in [6] for the development of a single, formal design methodology that supports designers in analysing the performance of protocols, with a semi-mechanizable procedure. We provide a general framework, where quantitative aspects, symbolically represented by parameters, can be formally estimated. By changing only these parameters on the architecture and the algorithm chosen, one can compare different implementations of the same protocol or different protocols. This allows the designer to choose among different alternatives, based on an evaluation of the trade-off between security guarantees and their price.

* Supported in part by the EU IST-2001-32072 project DEGAS.

We are mainly interested in evaluating the cost of each cryptographic operation and of each message exchange. Here, "cost" means any measure of quantitative properties such as speed, availability, etc.

Usually protocols are described through informal narrations. These narrations include only a list of the messages to be exchanged, leaving it unspecified which are the actions to be performed in receiving these messages (inputs, decryptions and possible checks on them). This can lead, in general, to an inaccurate estimation of costs. The above motivates the choice of using the process algebra LySA [3, 5], a close relative of the π- [24] and Spi-calculus [1], that details the protocol narration, in that outputs and the corresponding inputs are made explicit and similarly for encryptions and the corresponding decryptions. Also, LySA is explicit about which keys are fresh and about which checks are to be performed on the received values. More generally, LySA provides us with a unifying framework, in which security protocols can be specified and statically analysed [3, 5] through Control Flow Analysis. This analysis, fully automatic and always terminating, is strong enough to report known flaws on a wide range of protocols, and even to find new ones [4].

Technically, we give LySA (Sect. 2) an enhanced semantics, following [14], and then we associate rates to each transition, in the style of [26]. It suffices to have information about the activities performed by the components of a system in isolation, and about some features of the network architecture. We then mechanically derive Markov chains using these rates (Sect. 3). The actual performance evaluation is carried out using standard techniques and tools [33, 31, 32]. Significantly, quantitative measures, typically on cryptography, here live together with the usual qualitative semantics, where instead these aspects are usually abstracted away. Specifically, there exists a very early prototype, based on π-calculus, on which it is possible to run LySA, that we used for the case study presented here (Sect. 4), along with a standard mathematical tool such as Mathematica. Relative approaches are EMPA[8] and PEPA[19], to cite only a few.

In comparing different versions of the same protocol or different protocols, specified in LySA, our technique can be suitably integrated with the Control Flow one, to check security at the same stage.

Our framework can be extended [7] to estimate the cost of security attacks. The typical capabilities of the Dolev-Yao attacker [16] go beyond the ones a legitimate principal has. The needed model includes a set of the possible extra actions in which the attacker exploits its computational power and its capability of guessing (see also [10] and [23]). It would be

interesting to deal with timing attacks as well, even though this may considerably complicate our model.

2 LySA and its Enhanced Semantics

The LySA calculus [3, 5] is based on the π- [24] and Spi-calculus [1], but differs from these essentially in two aspects: (i) the absence of channels: there is only one global communication medium to which all processes have access; (ii) the tests associated with input and decryption are naturally expressed using pattern matching. Below, we assume that the reader is familiar with the basics of process calculi.

Syntax The syntax consists of terms $E \in \mathcal{E}$ and processes $P \in \mathcal{P}$,

$$E ::= a \mid x \mid \{E_1, \cdots, E_k\}_{E_0}$$
$$P ::= \mathsf{0} \mid out.P \mid in.P \mid P_1 \mid P_2 \mid (\nu\, a)P \mid dec \text{ in } P \mid A(y_1, \ldots, y_n)$$

where we introduced the following abbreviations: • $out \triangleq \langle E_1, \cdots, E_k \rangle$, • $in \triangleq (E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_k)$, • $dec \triangleq \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}$. Intuitively, the process $\mathsf{0}$ or *nil* represents the null inactive process. The operator | describes parallel composition of processes. The operator (νa) acts as a static declaration for the name a in the process P the restriction prefixes. Restriction is therefore used to create new names such as nonces or keys. The process $\langle E_1, \cdots, E_k \rangle.P$ sends $E_1, \cdots, E_k$ on the net and then continues like P. The process $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k).P$ receives the tuple $E'_1, \cdots, E'_k$ and continues as $P[E_{j+1}/x_{j+1}, \ldots, E_k/x_k]$, provided that $E_i = E'_i$ for all $i \in [1, j]$. The intuition is that the matching succeeds when the first j values E'_i pairwise correspond to the values E_i, and the effect is to bind the remaining $k-j$ values to the variables $x_{j+1}, \cdots, x_k$. Note that, syntactically, a semi-colon separates the components where matching is performed from those where only binding takes place. The same simple form of patterns is also used for decryption (see [9] for a more flexible choice). In fact, the process $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k\}_{E_0}\ \mathsf{in}\ P$ decrypts $E = \{E'_1, \cdots, E'_k\}_{E'_0}$ with the key E_0. Whenever $E_i = E'_i$ for all $i \in [0, j]$, the process behaves as $P[E_{j+1}/x_{j+1}, \ldots, E_k/x_k]$. Finally, an agent is a static definition of a parameterised process. Each agent identifier A has a unique defining equation of the form $A(\tilde{y}) = P$, where $\tilde{y}$ denotes a tuple $y_1, \ldots, y_n$ of distinct names occurring free in P.

Working Example Consider the following basic Kerberos key agreement protocol [22] that is part of our case study. We assume that the AES algorithm [12] is the crypto-system used here.

$$
\begin{array}{ll}
 & 1.\ A \rightarrow S : A, B \\
 & 2.\ S \rightarrow A : \{B, T, L, K_{AB}\}_{K_A}, \{A, T, L, K_{AB}\}_{K_B} \\
(Kerberos) & 3.\ A \rightarrow B : \{A, T, L, K_{AB}\}_{K_B}, \{A, T\}_{K_{AB}} \\
 & 4.\ B \rightarrow A : \{T, T\}_{K_{AB}}
\end{array}
$$

Intuitively, principal A asks the Key Distribution Center S for a session key to share with B. S generates the key K_{AB}, a timestamp T and lifetime L and produces an encryption of these components for A and another one for B, including the identity of the other principal. Both encryptions are sent to A, that can decrypt the first and forward the second to B, along with another encryption that A obtains by encoding (A, T) with the new key. B can decrypt the first encryption so to obtain K_{AB} then B decrypts the second encryption, and uses K_{AB} to encrypt (T, T) as a replay to A. To simplify, we use $\{T, T\}_{K_{AB}}$ rather than the usual $\{T + 1\}_{K_{AB}}$.

1 $Sys_1 = (\nu K_A)(\nu K_B)((A \mid B) \mid S)$	K_A, K_B long-term keys
2 $A = (\langle A, B \rangle . A')$	A sends msg (1)
4 $A' = (; v^A_{enc}, v^B_{enc}) . A''$	A receives and checks msg (2)
5 $A'' = \mathsf{decrypt}\ v^A_{enc}\ \mathsf{as}\ \{B; v_T, v_L, v_K\}_{K_A}\ \mathsf{in}\ A'''$	A decrypts the enc in msg (2)
6 $A''' = \langle v^B_{enc}, \{A, v_T\}_{v_K} \rangle . A''''$	A sends msg (3)
7 $A'''' = (; w^A_{enc}) . A'''''$	A receives and checks msg (4)
8 $A''''' = \mathsf{decrypt}\ w^A_{enc}\ \mathsf{as}\ \{v_T, v_T; \}_{v_K}\ \mathsf{in}\ A$	A decrypts the enc in msg (4)
9 $B = (; z^1_{enc}, z^2_{enc}) . B'$	B receives and checks msg (3)
10 $B' = \mathsf{decrypt}\ z^1_{enc}\ \mathsf{as}\ \{; z_A, z_T, z_L, z_K\}_{K_B}\ \mathsf{in}\ B''$	B decrypts the 1^{st} enc in msg (3)
11 $B'' = \mathsf{decrypt}\ z^2_{enc}\ \mathsf{as}\ \{z_A, z_T; \}_{z_K}\ \mathsf{in}\ B'''$	B decrypts the 2^{nd} enc in msg (3)
12 $B''' = \langle \{z_T, z_T\}_{z_K} \rangle . B$	B sends msg (4)
13 $S = (; y^A, y^B) . S'$	S receives and checks msg (1)
14 $S' = (\nu K_{AB})(\nu T)(\nu L)$	K_{AB} fresh session key
15 $(\langle \{y^B, T, L, K_{AB}\}_{K_A}, \{y^A, T, L, K_{AB}\}_{K_B} \rangle . S)$	S sends msg (2)

Table 1. Specification of *Kerberos* Protocol

The protocol specification in LySA is in Tab. 1, where the right column reports a concise explanation of the action on the left, in terms of the number of the message (called *msg*, while *enc* stands for an encrypted term) in the protocol narration. The whole system is given by the parallel composition ($\mid$) of the three processes A, B, S. Each part of the system performs a certain number of actions and then restarts.

Enhanced Operational Semantics. Here, we give a concrete version of operational semantics, called *enhanced* in the style of [13, 14]. Our enhanced semantics for LySA is a reduction semantics, built on top of the stan-

dard reduction semantics [3], where both processes and transitions are annotated with labels that will be helpful for computing costs.

Formally, each transition is enriched with an *enhanced label* θ which records both the action corresponding to the transition and its syntactic context. Actually, the label of a communication transition records the two actions (input and output) that lead to the transition. To facilitate the definition of our reduction semantics, for each given process, we annotate each of its sub-processes P with an encoding of the context in which P occurs. The encoding is a string of tags ϑ, that essentially record the syntactic position of P w.r.t. the parallel composition nesting. To do this, we exploit the abstract syntax tree of processes, built using the binary parallel composition as operator. We introduce a tag $\|_0$ ($\|_1$, resp.) for the left (for the right, resp.) branch of a parallel composition. Labels are defined as follows.

Definition 1. *Let* $\mathcal{L} = \{\|_0, \|_1\}$. *Then, the set of* context labels *is defined as* $\mathcal{L}^*$, *i.e. the set of all the string generated by* $\mathcal{L}$, *ranged over by* ϑ.

We choose to have tags concerned with the parallel structure of processes, i.e. linked to parallel composition "|". For our present purpose, this is the only necessary annotation (for other annotations, see [26, 14]).

Technically, labelled processes are inductively obtained in a pre processing step, by using the function $\mathcal{T}$. This function (inductively) prefixes actions with context labels: $\mathcal{T}$ unwinds the syntactic structure of processes, until reaching a $\mathsf{0}$ or a constant. Given a process P, this transformation operates in linear time with the number of prefixes. Note that this pre-processing step can be completely mechanized. An auxiliary function $\triangleright$ is needed to distribute context labels on processes.

Definition 2. *Let* $\mathcal{LP}$ *be the set of* Labelled Processes, *ranged over by* T, T', T_0, T_1. *The functions* $\mathcal{T} : \mathcal{P} \to \mathcal{LP}$ *and* $\triangleright$: $\mathcal{L}^* \times \mathcal{LP} \to \mathcal{LP}$, *written as* $\vartheta \triangleright T$, *are defined by induction in the box below:*

$-\ \mathcal{T}(\mathsf{0}) = \mathsf{0}$	$-\ \vartheta \triangleright \mathsf{0} = \mathsf{0}$
$-\ \mathcal{T}(\mu.P) = \mu.\mathcal{T}(P), \quad \mu \in \{out, in\}$	$-\ \vartheta \triangleright (\vartheta'\mu.T) = \vartheta\vartheta'\mu.(\vartheta \triangleright T), \quad \mu \in \{out, in\}$
$-\ \mathcal{T}(P_0 \mid P_1) = \|_0 \triangleright \mathcal{T}(P_0) \mid \|_1 \triangleright \mathcal{T}(P_1)$	$-\ \vartheta \triangleright (T_0 \mid T_1) = (\vartheta \triangleright T_0) \mid (\vartheta \triangleright T_1)$
$-\ \mathcal{T}((\nu a)P) = (\nu a)\mathcal{T}(P)$	$-\ \vartheta \triangleright (\nu a)T = (\nu a)\ \vartheta \triangleright T$
$-\ \mathcal{T}(A(y_1, \ldots, y_n)) = A(y_1, \ldots, y_n)$	$-\ \vartheta \triangleright \vartheta' A(y_1, \ldots, y_n) = \vartheta\vartheta' A(y_1, \ldots, y_n)$
$-\ \mathcal{T}(dec \ \mathsf{in}\ P) = dec \ \mathsf{in}\ \mathcal{T}(P)$	$-\ \vartheta \triangleright \vartheta'\ dec \ \ \mathsf{in}\ T = \vartheta\vartheta'\ dec \ \mathsf{in}\ (\vartheta \triangleright T)$

The following example illustrates how $\mathcal{T}$ works on the process $Sys_1 = ((A \mid B) \mid S)$. The context labels preceding the prefixes of the three processes are: $\vartheta_A = \|_0\|_0$ for A, $\vartheta_B = \|_0\|_1$ for B, and $\vartheta_S = \|_1$ for S.

$$\mathcal{T}(((A \mid B) \mid S)) = \|_{\mathbf{0}} \triangleright (\mathcal{T}(A \mid B))|||_{\mathbf{1}} \triangleright \mathcal{T}(S) =$$

$$\|_{\mathbf{0}} \triangleright (\|_{\mathbf{0}} \triangleright (\mathcal{T}(A)|||_{\mathbf{1}} \triangleright (\mathcal{T}(B))|||_{\mathbf{1}} \triangleright \mathcal{T}(S) = (\|_{\mathbf{0}}\|_{\mathbf{0}} \triangleright (\mathcal{T}(A)|||_{\mathbf{0}}\|_{\mathbf{1}} \triangleright (\mathcal{T}(B))|||_{\mathbf{1}} \triangleright \mathcal{T}(S)$$

For instance B is annotated with the label $\vartheta = \|_0\|_1$ as B is inside the right branch of the inner parallel composition $(A \mid B)$, and in turn on the left branch of the outermost parallel composition in $((A \mid B) \mid S)$.

The *enhanced label* of a transition records its *action*, i.e. decryption or input and output communications that lead to the transition. Also, actions come prefixed by their context labels.

Definition 3. *The set* $\Theta \ni \theta, \vartheta_O, \vartheta_I$ *of* enhanced labels *is defined by*

$$\theta ::= \langle \vartheta_O \; out, \vartheta_I \; in \rangle \mid \langle \vartheta \; dec \rangle$$

As usual, our semantics consists of the standard structural congruence $\equiv$ on processes and of a set of rules defining the transition relation.

Our *reduction relation* $\xrightarrow{\theta} \subseteq \mathcal{LP} \times \mathcal{LP}$ is the least relation on closed labelled processes that satisfies the rules in Tab. 2. In the rule (Com), the context labels ϑ_O (and ϑ_I, resp.) of both the partners are recorded in the pair $\langle \vartheta_O out, \vartheta_I in \rangle$ together with the corresponding output (and input, resp.) prefix. In the rule for decryption, the context label ϑ is recorded together with *dec* in the label of the transition. The other rules are quite standard. Our semantics differs from the standard one [3] because (i) processes are enriched with context labels ϑ and (ii) reductions carry enhanced labels θ. By eliminating labels from both transitions and processes, it is possible to recover the original reduction semantics $\longrightarrow \subseteq \mathcal{P} \times \mathcal{P}$.

For technical reasons, hereafter, we will restrict ourselves to *finite* state processes, i.e. whose corresponding transition systems have a finite set of states. Note that this does not mean that the behaviour of such processes is finite, because their transition systems may have loops.

Example (cont'd) In Fig. 1, we present the (finite) transition systems corresponding to Sys_1. To improve readability, we add a further component to the labels ϑ_i of transitions. A transition from state T to state T' has the form $T \xrightarrow{(\theta, caption)} T'$, where *caption* is a concise description of the step of the protocol narration. More precisely, it refers to message exchanges and decryptions (abbreviated as *dec*). Captions are of no other use.

The enhanced labels of Sys_1 are reported below. Since in our example, transitions have different labels each, we feel free to use hereafter the label θ_i for the i-th transition.

(Com)

$$\frac{\wedge_{i=1}^{j} E_i = E'_i}{\vartheta_O\ out.T \ \mid\ \vartheta_I\ in.T' \xrightarrow{\langle \vartheta_O out, \vartheta_I in\rangle} T \ \mid\ T'[E_{j+1}/x_{j+1}, \cdots, E_k/x_k]}$$

(Decr)

$$\frac{\wedge_{i=0}^{j} E_i = E'_i}{\vartheta\ dec \text{ in } T \xrightarrow{\langle \vartheta\ dec\rangle} T[E_{j+1}/x_{j+1}, \cdots, E_k/x_k]}$$

(Par)

$$\frac{T_0 \xrightarrow{\theta} T'_0}{T_0 \mid T_1 \xrightarrow{\theta} T'_0 \mid T_1}$$

(Res)

$$\frac{T \xrightarrow{\theta} T'}{(\nu\, a)T \xrightarrow{\theta} (\nu\, a)T'}$$

(Ide) :

$$\frac{\mathcal{T}(P)\{\tilde{K}/\tilde{y}\} \xrightarrow{\theta} T'}{\vartheta A(\tilde{K}) \xrightarrow{\vartheta\theta} \vartheta \rhd T'}, \quad A(\tilde{y}) = P$$

(Congr)

$$\frac{T \equiv T_0 \ \wedge\ T_0 \xrightarrow{\theta} T_1 \ \wedge\ T_1 \equiv T'}{T \xrightarrow{\theta} T'}$$

$out = \langle E_1, \cdots, E_k\rangle$,
$in = (E'_1, \cdots, E'_j; x_{j+1}, \cdots, x_k)$,
$dec = \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}$

Table 2. Enhanced Reduction Semantics, $T \xrightarrow{\theta} T'$.

$\theta_0 = \langle \vartheta_A\langle A, B\rangle, \vartheta_S(; z^1_{enc}, z^2_{enc})\rangle$
$\theta_1 = \langle \vartheta_S\langle \{y^B, T, L, K_{AB}\}_{K_A}, \{y^A, T, L, K_{AB}\}_{K_B}\rangle, \vartheta_A(; v^A_{enc}, v^B_{enc})\rangle$
$\theta_2 = \langle \vartheta_A\ \mathsf{decrypt}\ \{B, T, L, K_{AB}\}_{K_A}\ \mathsf{as}\ \{B; v_T, v_L, v_K\}_{K_A}\rangle$
$\theta_3 = \langle \vartheta_A\langle \{A, T, L, K_{AB}\}_{K_B}, \{A, T\}_{K_{AB}}\rangle, \vartheta_B(; z^1_{enc}, z^2_{enc})\rangle$
$\theta_4 = \langle \vartheta_B\ \mathsf{decrypt}\ \{A, T, L, K_{AB}\}_{K_B}\ \mathsf{as}\ \{; z_A, z_T, z_L, z_K\}_{K_B}\rangle$
$\theta_5 = \langle \vartheta_B\ \mathsf{decrypt}\ \{A, T\}_{K_{AB}}\ \mathsf{as}\ \{A, T; \}_{K_{AB}}\rangle$
$\theta_6 = \langle \vartheta_B\langle \{T, T\}_{K_{AB}}\rangle, \vartheta_A(; w^A_{enc})\rangle$
$\theta_7 = \langle \vartheta_A\ \mathsf{decrypt}\ \{T, T\}_{K_{AB}}\ \mathsf{as}\ \{T, T; \}_{K_B}\rangle$

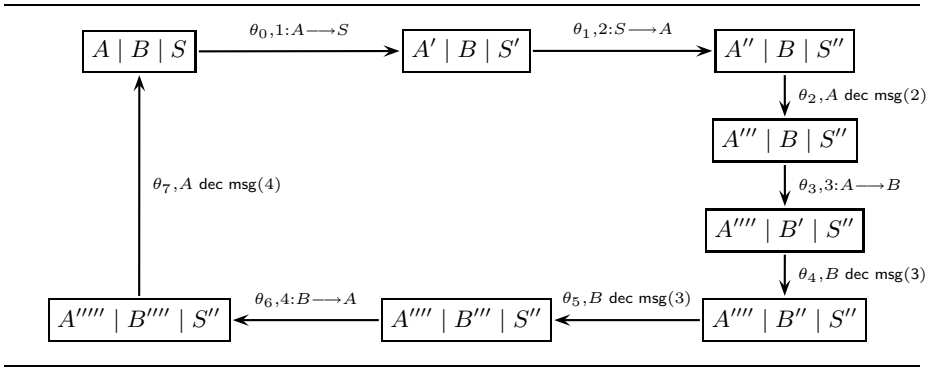

Fig. 1. Sys_1 Transition System.

3 Stochastic Analysis

Costs of transitions are derived by inspecting enhanced labels, following [26]. This information is sufficient to extract the necessary quantitative information to obtain the Continuous Time Markov Chains (CTMC) (see [2, 25] for more details on the theory of stochastic processes). In general, by "cost" we mean any measure that affects quantitative properties of transitions: here, we intend the time the system is likely to remain within a given transition. We specify the cost of a protocol in terms of the time overhead due to its primitives (along the same lines as [28]). The cost of (the component of) the transition depends on both the current action and on its context. Since the semantics of a language specifies its abstract machine, the context in which an action occurs represents the run-time support routines that the target machine performs to fire that action.

First, we intuitively present the main factors that influence the costs of actions and those due to their context. For simplicity, here we ignore the costs for other primitives, e.g. restriction or constant invocation (see [26] for a complete treatment).

- The cost of a *communication* depends on the costs of the input and output components. In particular, the cost of an (i) *output* depends on the size of the message and on the cost of each term of the message sent, in particular on its encryptions; (ii) *input* depends on the size of the message and on the cost of checks needed to accept the message. Actually, the two partners independently perform some low-level operations locally to their environment, each of which leads to a delay. Since communication is synchronous and handshaking, the overall cost corresponds to the cost paid by the slower partner.
- The cost of both *encryption* and *decryption* depends on the sizes of the cleartext and ciphertext, resp.; the complexity of the algorithm that implements it; the cipher mode adopted; the kind of the key (short/long, short-term/long-term). The length of the key is important: usually, the longer the key, the greater the computing time. In addition, the cost for *decryption* depends on the cost of the checks needed to accept the decryption.
- The cost of *parallel composition* is evaluated according to the number of available processors and to the speed of system clock.

To define a cost function, we start by considering the execution of each action on a dedicated architecture that only has to perform that action, and we estimate the corresponding duration with a fixed rate r. Then we model the performance degradation due to the run-time support. To do

that, we introduce a scaling factor for r in correspondence with each routine called by the implementation of the transition θ under consideration. Here, we just propose a format for these functions, with parameters to be instantiated on need. Note that these parameters depend on the target machine, e.g. in a system where the cryptographic operations are performed at very high speed (e.g. by a cryptographic accelerator), but with a slow link (low bandwidth), the time will be low for encryptions and high for communication; vice versa, in a system offering a high bandwidth, but poor cryptography resources.

Technically, we interpret costs as parameters of exponential distributions $F(t) = 1 - e^{-rt}$, with rate r and t as time parameter (general distributions are also possible see [30]). The *rate* r associated with the transition is the parameter which identifies the exponential distribution of the duration times of the transition, as usual in stochastic process algebras (e.g. [19, 18]). The shape of $F(t)$ is a curve which grows from 0 asymptotically approaching 1 for positive values of its argument t. The parameter r determines the slope of the curve: the greater r, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time x is $F(x) = 1 - e^{-rx}$, so r determines the time, Δt, needed to have a probability near to 1.

3.1 Cost Functions

We define in a few steps the function that associates rates with communication and decryption transitions, or, more precisely, with their enhanced labels. We first give the auxiliary function $f_E : \mathcal{E} \to \mathbb{R}^+$ that estimates the effort needed to manipulate terms $E \in \mathcal{E}$.

- $f_E(a) = size(a)$
- $f_E(\{E_1, \ldots, E_k\}_{E_0}) = f_{enc}(f_E(E_1), ..., f_E(E_1), kind(E_0))$

The size of a name a ($size(a)$) matters. For an encrypted term, we use the function f_{enc}, which in turn depends on the estimate of the terms to encrypt and on the kind of the key (represented by $kind(E_0)$), i.e. on its length and on the corresponding crypto-system.

Then we assign costs to actions in $\{in, out, dec\}$. Formally, the function $\$_\alpha : \{in, out, dec\} \to \mathbb{R}^+$ is defined as

- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = f_{out}(f_E(E_1), ..., f_E(E_1), bw)$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = f_{in}(f_E(E_1), ..., f_E(E_j), match(j), bw)$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}) =$
  $f_{dec}(f_E(E), kind(E_0), match(j))$

The functions f_{out} and f_{in} define the costs of the routines which implement the send and receive primitives. Besides the implementation cost

due to their own algorithms, the functions above depend on the bandwidth of the communication channel (represented by bw) and the cost of the exchanged terms, in turn computed by the auxiliary function f_E. Also, the cost of an input depends on the number of tests or matchings required (represented by $match(j)$). Finally, the function f_{dec} represents the cost of a decryption. It depends on the manipulated terms ($f_E(E)$), on the kind of key ($kind(E_0)$) and on the number of matchings ($match(j)$).

We now consider the context in which the actions occur. To determine the slowing factor due to parallel composition, we associate a cost to each context label ϑ, as expressed by the function $\$_l : \{\|_0, \|_1\}^* \to (0, 1]$. Parallel composition is evaluated according to the number np of processors available, and on the number of processes that run on them. Another factor is given by the speed of $clock$, the system clock.

- $\$_l(\vartheta) = f_{||}(np, |\vartheta|, clock)$

Finally, the function $\$: \Theta \to \mathbb{R}^+$ associates rates with enhanced labels.

- $\$(\langle \vartheta_O out, \vartheta_I in \rangle) = \min\{\$_l(\vartheta_O) \cdot \$_\alpha(out), \$_l(\vartheta_I) \cdot \$_\alpha(in)\}$
- $\$\langle \vartheta dec \rangle = \$_l(\vartheta) \cdot \$_\alpha(dec)$

As mentioned above, the two partners independently perform some low-level operations locally to their environment, represented by the two context labels ϑ_O and ϑ_I. Each label leads to a delay ($\$_l(\vartheta_O)$ and $\$_l(\vartheta_I)$, resp.) in the rate of the corresponding action ($\$_\alpha(out)$ and $\$_\alpha(in)$, resp.). Thus, the cost paid by the slower partner corresponds to the minimum cost of the operations performed by the participants, in isolation. Indeed the lower the cost, i.e. the rate, the greater the time needed to complete an action and hence the slower the speed of the transition occurring. The smaller r, the slower $F(t) = 1 - e^{-rt}$ approaches its asymptotic value.

Note that we do not fix the actual cost function: we only propose for it a set of parameters to reflect some features of an idealized architecture and of a particular cryptosystem. Although very abstract, this suffices to make our point. A precise instantiation comes with the refinement steps from specification to implementations as soon as actual parameters become available.

We now associate a rate to each transition in the transition system Sys_1. For the sake of simplicity, we assume that each principal has enough processing power and then we can map each ϑ to 1. We could vary this value considering e.g. differences in the speed of clock for the two processes. We instantiate the cost functions given above, by using the following parameters each used to compute the rate corresponding to a particular action (sending, receiving and decryption) or a part of it, such as an encryption or a pattern matching: (i) `e` and `d` for encrypting and for

decrypting, (*ii*) $\mathtt{s}$ and $\mathtt{r}$ for sending and for receiving, (*iii*) $\mathtt{m}$ for pattern matching. The functions are:

- $f_E(a) = 1$
- $f_E(\{E_1, \ldots, E_k\}_{E_0}) = \frac{\mathtt{e}}{\mathtt{s}} \cdot \sum_{i=1}^{k} f_E(E_i) + \sum_{i=1}^{k} f_E(E_i)$
- $\$_\alpha(\langle E_1, \ldots, E_k \rangle) = \frac{1}{\mathtt{s} \cdot \sum_{i=1}^{i} f_E(E_i)}$
- $\$_\alpha((E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)) = \frac{1}{\mathtt{r} \cdot k + \mathtt{m} \cdot j}$
- $\$_\alpha(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j; x_{j+1}, \cdots, x_k\}_{E_0}) = \frac{1}{\mathtt{d} \cdot k + \mathtt{m} \cdot j}$

Intuitively, these parameters represent the time spent performing the corresponding action on a single term. They occur in the denominator, therefore keeping the rule that the faster the time, the slower the rate. Since transmission is usually more time-consuming than the corresponding reception, the rate of a communication, will always be that of output.

Example (cont'd) The rate c_0 of the first transition of Sys_1 is $\frac{1}{2\mathtt{s}}$:
$c_0 = \$(\theta_0) = min\{(\$_l(\vartheta_A) \cdot \$_\alpha(\langle A, B \rangle, \$_l(\vartheta_S) \cdot \$_\alpha((; z^1_{enc}, z^2_{enc}))\} = min\{\frac{1}{2\mathtt{s}}, \frac{1}{2\mathtt{r}}\}$.
All the rates $c_i = \$(\theta_i)$ are: $c_0 = \frac{1}{2\mathtt{s}}$, $c_1 = \frac{1}{8\mathtt{s}+8\mathtt{e}}$, $c_2 = \frac{1}{4\mathtt{d}+\mathtt{m}}$, $c_3 = \frac{1}{6\mathtt{s}+6\mathtt{e}}$, $c_4 = \frac{1}{4\mathtt{d}+\mathtt{m}}$ $c_5 = \frac{1}{2\mathtt{d}+2\mathtt{m}}$, $c_6 = \frac{1}{2\mathtt{s}+2\mathtt{e}}$ and $c_7 = \frac{1}{2\mathtt{d}+2\mathtt{m}}$.

3.2 Markov Chains and Performance Measures

Our first step is obtaining a Continuous Time Markov Chain (CTMC) from a transition system. Then, we shall calculate the actual performance measure, e.g. the throughput or utilization of a certain resource. We use the rates of transitions computed in Subsection 3.1, to transform a transition system T into its corresponding $CTMC(T)$: a state is associated with each node of the transition system, while the transitions between states are defined by the arcs.

Actually, the rate $q(T_i, T_j)$ at which a system changes from behaving like process T_i to behaving like T_j is the sum of the single rates of all the possible transitions from T_i to T_j. Note that $q(T_i, T_j)$ coincides with the off-diagonal element q_{ij} of the generator matrix of the CTMC, namely $\mathbf{Q}$. Recall that a CTMC can be seen as a directed graph and that its matrix $\mathbf{Q}$ (apart from its diagonal) represents its adjacency matrix. Hence, hereafter we will use indistinguishably CTMC and its corresponding $\mathbf{Q}$ to denote a Markov chain. More formally, the entries of the generator matrix $\mathbf{Q}$ are defined as $q_{ij} = \begin{cases} q(T_i, T_j) = \sum\limits_{T_i \xrightarrow{\theta_k} T_j} \$(\theta_k) \text{ if } i \neq j \\ -\sum\limits_{j=0, j \neq i}^{n} q_{ij} \qquad \text{if } i = j \end{cases}$

Example (cont'd) Consider the transition system Sys_1. Since it is finite and has a cyclic initial state, then there exists its stationary distribution.

The *stationary probability distribution* of a CTMC is $\Pi = (X_0, \ldots, X_{n-1})$ such that Π solves the matrix equation $\Pi^T \mathbf{Q} = \mathbf{0}$ and $\sum_{i=0}^{n} X_i = 1$. We derive the following generator matrix $\mathbf{Q_1}$ of $CTMC(Sys_1)$ and the corresponding stationary distributions is Π_1, where $C = 9\mathtt{s}+8\mathtt{e}+6\mathtt{d}+3\mathtt{m}$.

$$\mathbf{Q_1} = \begin{bmatrix} -c_0 & c_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -c_1 & c_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -c_2 & c_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -c_3 & c_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -c_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -c_5 & c_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -c_6 & c_6 \\ c_7 & 0 & 0 & 0 & 0 & 0 & 0 & -c_7 \end{bmatrix}$$

$$\Pi_1 = \left[\frac{\mathtt{s}}{\mathtt{C}}, \frac{4(\mathtt{s}+\mathtt{e})}{\mathtt{C}}, \frac{4\mathtt{d}+\mathtt{m}}{\mathtt{C}}, \frac{3(\mathtt{s}+\mathtt{e})}{\mathtt{C}}, \frac{4\mathtt{d}+\mathtt{m}}{\mathtt{C}}, \frac{\mathtt{d}+\mathtt{m}}{\mathtt{C}}, \frac{\mathtt{e}+\mathtt{s}}{\mathtt{C}}, \frac{\mathtt{d}+\mathtt{m}}{\mathtt{C}}, \right]$$

Evaluating the Performance In order to define performance measures for a process T, we define a *reward structure* associated with T, following [21, 19, 11]. Usually, a reward structure is simply a function that associates a reward with any state passed through in a computation of T. For instance, when calculating the utilisation of a resource, we assign value 1 to any state in which the use of the resource is enabled (typically the source of a transition that uses the resource). All the other states earn the value 0. Instead we use a slightly different notion, where rewards are computed from rates of transitions [26]. To measure instead the throughput of a system, i.e. the amount of useful work accomplished per unit time, a reasonable choice is to use as nonzero reward a value equal to the rate of the corresponding transition. The reward structure of a process T is a vector of rewards with as many elements as the number of states of T. By looking at the stationary distribution of and varying the reward structure, we can compute different performance measures. The total reward is obtained by multiplying the stationary distribution and the reward structure.

Definition 4. *Given a process T, let $\Pi = (X_0, \ldots, X_{n-1})$ be its stationary distribution and $\rho = \rho(0), ..., \rho(n-1)$ be its reward structure. The* total reward *of T is computed as $R(T) = \sum_i \rho(i) \cdot X_i$.*

Example (cont'd) The throughput for a given activity is found by first associating a transition reward equal to the activity rate with each transition. In our systems each transition is fired only once. Also, the graph of the corresponding CTMC is cyclic and all the labels represent different activities. This amounts to saying that the throughput of all the activities is the same, and we can freely choose one of them to compute the

throughput of Sys_1. Thus we associate a transition reward equal to its rate with the last communication and a null transition reward with all the others communications. From them, we compute the reward structures as $\rho_1 = (0, 0, 0, 0, 0, 0, c_7)$, where $c_7 = \frac{1}{2\mathtt{d}+2\mathtt{m}}$. The total reward $R(Sys_1)$ of the system amounts then to $\frac{\mathtt{d}+\mathtt{m}}{(2\mathtt{d}+2\mathtt{m})(9\mathtt{s}14+\mathtt{d}+3\mathtt{m})}$. To use this measure, it is necessary to instantiate our parameters under various hypotheses, depending on several factors, such as the network load, the packet size, the number of mobile stations and so on. We delay this kind of considerations to the next section, where this measure will be compared with the one obtained for a different protocol.

4 A Case Study

It is well known that asymmetric key cryptography is more expensive than symmetric key cryptography. This is why often the first technique is adopted for long-term keys, while the other is exploited for session keys. We want to apply our framework and compare public versus secret encryption techniques. Following [20], we compare the two key-agreement protocols Kerberos [22] (the one used as working example) and Diffie-Hellman, compared there for their energy consumption.

Before the comparison, we need to illustrate the second protocol and to apply it the introduced technique. The Diffie-Hellman protocol is based on the use of two functions, i.e. $g(x) = \alpha^x \ mod \ p$ and $f(x, y) = y^x \ mod \ p$, where p is a large prime (public) and α of Z_p^* (the set of all the numbers prime with p) is a generator. Here, we can safely abstract from the underlying number theory. We need to slightly extend the syntax with the following productions, where E and E' are terms and each of the two functions g and f are considered as term constructors. The semantics is modified accordingly, by adding a case for the function $\mathcal{T}$ and by adding an axiom to the reduction semantics.

$E ::= g(E) \mid f(E, E')$
$P ::= \mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ P$

$\mathcal{T}(\mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ P) = \mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ \mathcal{T}(P)$
$\vartheta \rhd \vartheta'\ (\mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ T) = \vartheta\vartheta'\ \mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ (\vartheta \rhd T)$
$\vartheta\ \mathsf{let}\ x\ \mathsf{be}\ f(E, E')\ \mathsf{in}\ T \xrightarrow{\langle \vartheta f \rangle} T[f(E, E')/x]$

The protocol is simply: (*Diffie-Hellman*) $\begin{array}{l} 1.\ A \rightarrow B : g(K_A) \\ 2.\ B \rightarrow A : g(K_B) \end{array}$

At the end of the exchange A computes the key as $f(K_A, g(K_B))$, while B computes it as $f(g(K_A), K_B)$. These steps are made explicit in the LySA specification of the protocol, $Sys_2 = (A|B)$, given in Tab. 3. The two keys

coincide, because of the following algebraic rule: $f(x, g(y)) = f(y, g(x))$. Here, K_A and K_B serve as private keys, while $g(K_A)$ and $g(K_B)$ serve as public keys. Note that here we do not need to extend our syntax with asymmetric encryption (we refer the reader to [3]).

1 $Sys_2 = (A \mid B)$	
2 $A = (\nu K_A)$	K_A private key
3 $(\langle g(K_A) \rangle . A')$	A sends msg (1)
4 $A' = (; v_g^A) . A''$	A receives msg (2)
5 $A'' = \mathsf{let}\ v_g^A\ \mathsf{be}\ f(K_A, v_g^A)\ \mathsf{in}\ A$	A computes f
6 $B = (\nu K_B)$	K_B private key
7 $B' = (; v_g^B) . B''$	B receives msg (1)
8 $(\langle g(K_B) \rangle . B')$	B sends msg (2)
9 $B'' = \mathsf{let}\ v_g^B\ \mathsf{be}\ f(K_B, v_g^A)\ \mathsf{in}\ B$	B computes f

Table 3. Specification of *Diffie-Hellman* Protocol

The Diffie-Hellman protocol can be efficiently implemented using the Elliptic Curve Asymmetric-key (ECC) algorithm [17], that operates over a group of points on an elliptic curve. For each curve a base point G is fixed, a large random integer k acts as a private key, while kG (*scalar point multiplication* that results in another point on the curve) acts as the corresponding public key. Scalar point multiplication is obtained by a combination of point-additions and point-doublings. So, we can use (1) $g(x) = xG$ and (2) $f(x, y) = xy$.

For lack of space, we omit here the (finite) transition systems of Sys_2, that like the one of Sys_1 has a unique cyclic path. We directly give the rates corresponding to the transitions:

$$c'_0 = \frac{1}{\mathtt{s}+4\mathtt{pm}}, \quad c'_1 = \frac{1}{\mathtt{s}+4\mathtt{pm}}, \quad c'_2 = \frac{1}{4\mathtt{pm}}, \quad c'_3 = \frac{1}{4\mathtt{pm}}.$$

We use the same cost parameters as in Section 3. In particular, we assume that the sending parameter $\mathtt{s}$ is the same used for Sys_1 (again transmission is more expensive than reception). Since the functions g and f in Sys_2 are both implemented with four elliptic curve point multiplications, we assume that the cost for g and f depend on the parameter $\mathtt{pm}$ (parameter for point multiplication), more precisely $\$_\alpha(f(E, E')) = \frac{1}{4\mathtt{pm}}$ and $\$_\alpha(\langle g(E) \rangle) = \frac{1}{\mathtt{s}+4\mathtt{pm}}$. Again, for the sake of simplicity, we assume that each principal has enough processing power, so $\$_|(\vartheta) = 1$ for each ϑ.

The stationary distribution Π_2, where $D = 2(8\mathtt{pm} + \mathtt{s})$, corresponding to $\mathbf{Q}_2$ of $CTMC(Sys_2)$, here omitted, is:

$$\Pi_2 = \left[\frac{\mathtt{s}+4\mathtt{pm}}{\mathtt{D}}, \ \frac{\mathtt{s}+4\mathtt{pm}}{\mathtt{D}}, \ \frac{4\mathtt{pm}}{\mathtt{D}}, \ \frac{4\mathtt{pm}}{\mathtt{D}}\right]$$

We can now compare the performance of the two protocols by relating their throughputs. As done before, we associate a transition reward equal to its rate with the last communication and a null transition reward with all the others communications. We compute then the reward structure $\rho_2 = (0, 0, 0, c'_3)$ for Sys_2 where $c'_3 = \frac{1}{4\mathtt{pm}}$. Furthermore, we assume the same cost for encryption and decryption, i.e. $\mathtt{e} = \mathtt{d}$. The total reward of Sys_2 is $R(Sys_2) = \frac{1}{2(\mathtt{s}+8\mathtt{pm})}$ and is such that:

$$R(Sys_1) - R(Sys_2) = \frac{8\mathtt{pm} - (8\mathtt{s} + 14\mathtt{d} + 3\mathtt{m})}{2((\mathtt{s} + 8\mathtt{pm}))((9\mathtt{s}14 + \mathtt{d} + 3\mathtt{m}))} > 0 \text{ if } \mathtt{pm} > \frac{(8\mathtt{s} + 14\mathtt{d} + 3\mathtt{m})}{8}$$

Experimentally, we know that point multiplication is significantly more time-consuming than decryption, therefore, we can assume that $\mathtt{pm}$ is significantly higher than $\mathtt{d}$. Consequently, we conclude that $R(Sys_1) > R(Sys_2)$, i.e. the first system has a better performance. Clearly, energy consumption of a cryptographic algorithm is strongly related to its time complexity and thus our result agrees with the one obtained in [20].

Actually, our working example presents a simple setting, in which the involved transition systems have a unique cyclic path. In general, transition systems have more loops. Typically, this happens with a multi-session version of the protocols presented before, where more copies of each principal (A, B and S) running in parallel, lead to more transitions with the same source. Also, this happens with non-repudiation protocols.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols - The Spi calculus. *Information and Computation*, 148(1):1–70, Jan 1999.
2. A. A. Allen. *Probability, Statistics and Queueing Theory with Computer Science Applications.* Academic Press, 1978.
3. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. *Proc. of CSFW'03*, pages 126–140. IEEE, 2003.
4. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control Flow Analysis can find new flaws too. *Proc. of Workshop on Issues in the Theory of Security (WITS'04)*, 2004.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocos. To appear in *Journal of Computer Securuty.*
6. C. Bodei, M. Buchholtz, M. Curti, P. Degano, F. Nielson, and H. Riis Nielson and C. Priami. Performance Evaluation of Security Protocols specified in Lysa. *Proc. of (QAPL'04)*, ENTCS 112, 2005.
7. C. Bodei, M. Curti, P. Degano, C. Priami. A Quantitative Study of Two Attacks. *Proc. of (WISP'04)*, ENTCS 121, 2005.
8. M. Bravetti, M. Bernardo and R. Gorrieri. Towards Performance Evaluation with General Distributions in Process Algebras.*Proc. of CONCUR98*, LNCS 1466, 1998.

9. M. Buchholtz, F. Nielson, and H. Riis Nielson. A calculus for control flow analysis of security protocols. *International Journal of Information Security*, 2 (3-4), 2004.
10. I. Cervesato Fine-Grained MSR Specifications for Quantitative Security Analysis. *Proc. of WITS'04*, pp. 111-127, 2004.
11. G. Clark. Formalising the specifications of rewards with PEPA. *Proc. of PAPM'96*, pp. 136-160. CLUT, Torino, 1996.
12. J. Daemen and V. Rijndael. *The design of Rijndael*. Springer-Verlag, 2002.
13. P. Degano and C. Priami. Non Interleaving Semantics for Mobile Processes. *Theoretical Computer Science*, 216:237–270, 1999.
14. P. Degano and C. Priami. Enhanced Operational Semantics. *ACM Computing Surveys*, 33, 2 (June 2001), 135-176.
15. W. Diffie and M. E. Hellman. New directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644-654, 1976.
16. D. Dolev and A. Yao. On the security of public key protocols. *IEEE TIT*, IT-29(12):198–208, 1983.
17. IEEE P1363 Standard Specification for Public-Key Cryptography, 1999
18. H. Hermanns and U. Herzog and V. Mertsiotakis. Stochastic process algebras – between LOTOS and Markov Chains. *Computer Networks and ISDN systems* 30(9-10):901-924, 1998.
19. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
20. A. Hodjat and I. Verbauwhede. *The Energy Cost of Secrets in Ad-hoc Networks.* IEEE Circuits and Systems Workshop on Wireless Communications and Networking, 2002.
21. R, Howard. *Dynamic Probabilistic Systems: Semi-Markov and Decision Systems.* Volume II, Wiley, 1971.
22. J.T. Kohl and B.C. Clifford. *The Kerberos network authentication service (V5).* The Internet Society, Sept. 1993.RCF 1510.
23. C. Meadows. A cost-based framework for analysis of denial of service in networks. *Journal of Computer Security*, 9(1/2), pp.143 - 164, 2001.
24. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Info. & Co.*, 100(1):1–77, 1992.
25. R. Nelson. *Probability, Stochastic Processes and Queeing Theory.* Springer, 1995.
26. C. Nottegar, C. Priami and P. Degano. Performance Evaluation of Mobile Processes via Abstract Machines. *Transactions on Software Engineering*, 27(10), 2001.
27. D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, 1987.
28. A. Perrig and D.Song. A First Step towards the Automatic Generation of Security Protocols. *Proc. of Network and Distributed System Security Symposium*, 2000.
29. G. Plotkin. A Structural Approach to Operational Semantics. *Tech. Rep. Aarhus University, Denmark*, 1981, DAIMI FN-19
30. C. Priami. Language-based Performance Prediction of Distributed and Mobile Systems *Information and Computation* 175: 119-145, 2002.
31. A. Reibnam and R. Smith and K. Trivedi. Markov and Markov reward model transient analysis: an overview of numerical approaches. *European Journal of Operations Research*: 40:257-267, 1989.
32. W. J. Stewart. *Introduction to the numerical solutions of Markov chains.* Princeton University Press, 1994.
33. K. S. Trivedi. *Probability and Statistics with Reliability, Queeing and Computer Science Applications.* Edgewood Cliffs, NY, 1982.