Titre: Title:	Using admissible interference to detect denial of service attacks
Auteurs: Authors:	Stéphane Lafrance, & John Mullins
Date:	2002
Type:	Rapport / Report
Référence: Citation:	Lafrance, S., & Mullins, J. (2002). Using admissible interference to detect denial of service attacks. (Technical Report n° EPM-RT-2002-01). https://publications.polymtl.ca/2595/

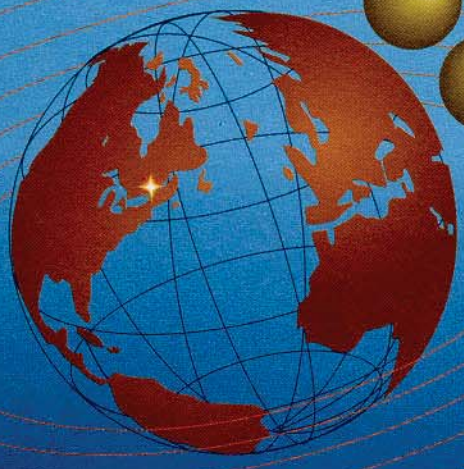
Institution:	École Polytechnique de Montréal
Numéro de rapport: Report number:	EPM-RT-2002-01
URL officiel: Official URL:	
Mention légale: Legal notice:	

EPM-RT-2002-01

Using Admissible Interference to Detect Denial of Service Attacks

Stéphane Lafrance, John Mullins
Département de génie informatique

Mai 2002



ÉCOLE POLYTECHNIQUE MONTRÉAL
Le génie sans frontières
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

EPM-RT-2002-01

Using Admissible Interference to Detect Denial of Service Attacks

Stéphane Lafrance, John Mullins
Département de génie informatique
École Polytechnique de Montréal

Mai 2002

Dépôt légal :
Bibliothèque nationale du Québec, 2002
Bibliothèque nationale du Canada, 2002

EPM-RT-2002-01
Using admissible interferrence to detect denial of service attacks
par : Stéphane Lafrance et John Mullins
Département de génie informatique.
École Polytechnique de Montréal

Using Admissible Interference to detect Denial of Service Attacks

Stéphane Lafrance* John Mullins†
Dept. of Computer Engineering
École Polytechnique‡
Campus of the University of Montreal

May 2002

Abstract

Recently, Meadows proposed a formal cost-based framework for the denial of service analysis. This paper operates within this framework and extends a method founded on bisimulation-based non-deterministic admissible interference, firstly intended to validate confidentiality and authenticity. Its contribution is to provide a model-checking method for a security property called *impassivity* that copes with robustness against denial of service attacks. More specifically, we introduce an information flow method having the capability to detect resource exhaustion attacks. An application of the method is given on the 1KP secure electronic payment protocol and leads to the detection of three denial of service attacks.

1 Introduction

Motivation In recent years, several Internet sites have been subjected to denial of service attacks. One of the most famous is the *SYN flooding* attack [25] on the TCP/IP protocol. Since 1996, this resource exhaustion attack has been launched at several occasions by intruders having the capabilities, with little effort, to initiate a large number of protocol runs. This denial of service on the TCP/IP protocol is possible due to the facility to forge fake identity, thus the difficulty for the victim to identify an attacker. Other denial of service attacks have been perpetrated on Internet e-commerce sites, including Yahoo, Ebay and E*trade in February 2000, and Microsoft in January 2001. A survey on this matter is given by Moore [19].

Formal methods to analyse security protocols Meanwhile, the sudden expansion of electronic commerce has introduced an urgent need to establish strong security policies for the design of security protocols. Formal validation of security protocols havs since became one of the primary task in computer science. Many methods, coming from a wide range of approaches, have been proposed in the literature to analyze security protocols including model-checking [1, 12, 15, 24], control flow analysis [5], logic programming [16] and rewriting and reachability problems [2, 23]. Most of these methods are devoted to validation of confidentiality and authentication policies. Little attention has been paid to denial of service up to now. But the lack of being able to clearly establish a formal definition for denial of service has made this type of attack a growing concern for protocols designers.

*Research supported by an FCAR doctoral scholarship (Quebec Government)
†Research supported by the NSERC grant no. 138321-01 from the Canadian Government
‡Mailing address: P.O. Box 6079, Station Centre-ville, Montreal (Quebec), Canada, H3C 3A7, {Stephane.Lafrance, John.Mullins}@polymtl.ca

The Meadow's framework to analyse robustness to denial of service attacks In network denial of service attacks on protocols for establishing authenticated communication channels, the identity of the attacker is generally unknown because authentication has not yet been completed. A design technique for making protocols more resistant to such attacks is the use of a sequence of authentication mechanisms, arranged in order of increasing cost (to both parties) and security. Thus, an attacker must be willing to complete the earlier stages of the protocol before it can force a system to expend resources running the later stages of the protocol. Recently, Meadows [17] has proposed a framework for analyzing such protocols. The framework is based on Gong and Syverson's model of fail-stop protocols [11]. Meadows interprets this fail-stop model by requirements specification based on Lamport's causally-precedes relation [14] that sets what events should causally-precede others in the protocol. The NRL protocol analyzer [16] was used to formally verify some properties of such protocols.

Admissible interference *Non-interference* properties [10] capture any causal dependency between private level actions and public level behavior which could be used to infer private information from public channels. However, many practical secrecy problems go beyond the scope of non-interference. As an example, cryptosystems permit classified or encrypted private information to flow safely onto unprotected (i.e. low-level) channels despite the obvious causal dependency between, on the one hand the secret data m and encryption key k, and, on the other hand the declassified data $\{m\}_k$ (m encrypted by k). Indeed, any variation of m or k is reflected in $\{m\}_k$. In this case, the basic concern is to ensure that programs leak sensitive information *only* through the cryptosystem or more generally, through the downgrading system. *Admissible interference* [20, 21] is such a property.

Application of admissible interference to detect potential denial of service attacks An admissible interference based method has been designed to analyse cryptographic protocols [12, 13]. The basic idea of the method is to prove that no intruder can interfere with the protocol unless the interference occurs through honest behaviors. This admissible interference can be expressed by simply identifying admissible attacks corresponding to harmless enemy actions occurring in the protocol. This paper introduces an admissible interference based security property and a bisimulation-based algorithm for validation of security protocols against denial of service. More specifically, we use a process algebra called *Security Protocols Process Algebra* (*SPPA*) [13], an extension of Milner's value-passing CCS [18], to specify the protocol and attackers as concurrent processes. Furthermore, every SPPA's action, including local function calls and information exchanges, is assigned a cost describing the quantity of resource used to execute it. Security protocols are validated against denial of service, more precisely resource exhaustion attacks, by verifying whether they satisfy an information flow property called *impassivity*. Impassivity requires that3 there be no causal dependency between low-cost enemy behaviors and high-cost behaviors of other principals. In other words, impassivity states that enemy actions do not cause interference on more costly actions. As an example, impassivity detects whenever an enemy process may use a protocol to send a fake message in order to force a costly action like decryption or signature checking. Such a flaw could be exploited with little effort by an attacker in order to launch a denial of service attack by wasting other principal's valuable resources.

Paper Organization The paper is organised as follows. Process algebra SPPA, that may cope with architecture of cryptographic protocols, is defined in Section 2. Admissible interference and its application to detection of potential denial of service attacks are presented in Sections 3 and 4 respectively. An application of our method to the 1KP secure electronic payment protocol [4, 3] is

developed in Section 5. This example leads to the discovery of potential denial of service attacks on this protocol.

2 A Process Algebra to Specify Security Protocols

The first step toward validation of security protocols is to find a language that may express both the protocols and the security policies we want to enforce. Process algebra have been used for some years to specify protocols as a cluster of concurrent processes, representing principals participating to the protocol, that are able to communicate in order to exchange data. CSP was one of the first process algebras to be successfully used in this matter [15, 24]. This paper uses a process algebra called SPPA [13], an extension of Milner's value-passing CCS [18] to cope with security protocols. SPPA allows the specification of local function calls and introduces marker actions that either come as responses or are mandatory to particular actions. First, we need to establish a framework for messages.

2.1 Message Algebra

We consider the following message algebra with the set of *terms*, ranged over by a and b, defined by the following grammar:

$$
\begin{array}{lll}
a, b, a_1, \ldots, a_n & ::= & \\
& \mid \quad v & (\textit{value}) \\
& \mid \quad x & (\textit{variable}) \\
& \mid \quad (a_1, \ldots, a_n) & (\textit{n-tuple}) \\
& \mid \quad \pi_i(a) & (\textit{projection}, \text{ where } a \text{ is a } n\text{-tuple and } i \leq n) \\
& \mid \quad \{a\}_b & (\textit{encryption}) \\
& \mid \quad [a]_b & (\textit{signature}) \\
& \mid \quad h(a) & (\textit{hash})
\end{array}
$$

For any term a, we denote $fv(a)$ the set of variables occurring in a and we say that a is a *closed term* whenever $fv(a) = \emptyset$. The set of all closed terms is denoted by $\mathcal{T}$. We consider a finite set $\mathcal{I}$ of values used as principal identifier, and a set $\mathcal{K} \subseteq \mathcal{T}$ of closed terms that may be used as encryption key. In order to deal with public-key encryption, we use an idempotent operator $[-]^{-1} : \mathcal{K} \to \mathcal{K}$ such that a^{-1} denotes the private decryption key corresponding to the public encryption key a, or vice versa. For symmetric encryption, we put $a^{-1} = a$.

2.2 Syntax of SPPA

We consider a finite set C of *public channels*. Commonly public channels are used to specify the exchange of messages between processes. Every public channel c has a predetermined domain $\mathrm{dom}(c)$ of closed terms that can be sent and received over c. We also consider a finite set Λ of *private functions* that range over terms and produce new terms using the grammar rules above (without introducing new variable). We write $\mathrm{dom}(\lambda)$ to denote the domain of closed terms of a private function λ. Whenever $a \notin dom(\lambda)$, we often write $\lambda(a) = \texttt{fail}$. We assume that every principal X has its own set Λ_X of private functions, such that $\Lambda = \bigcup_{X \in \mathcal{I}} \Lambda_X$. Intuitively, a principal X has only access to functions from Λ_X, which usually contains the following functions:

- $pair_X(a_1, \ldots, a_n) = (a_1, \ldots, a_n)$ (*pair function* with domain $\mathcal{T}^n$, for any n);
- $extract_X(i, a) = \pi_i(a)$ (*extraction function* with domain $\{i \in \mathbb{N} \mid i \leq n\} \times \{(a_1, \ldots, a_n) \mid a_1, \ldots, a_n \in \mathcal{T}\}$, for any n);

- $enc_X(k, a) = \{a\}_k$ (*encryption function* with domain $\{(k, a) \mid k \in \mathcal{K} \text{ and } a \in \mathcal{T}\}$);
- $dec_X(k^{-1}, \{a\}_k) = a$ (*decryption function* with domain $\{(k^{-1}, \{a\}_k) \mid k \in \mathcal{K} \text{ and } a \in \mathcal{T}\}$);
- $hash_X(a) = h(a)$ (*hash function* with domain $\mathcal{T}$);
- $sign_X(k, a) = [a]_k$ (*signature function* with domain $\{(k, a) \mid k \in \mathcal{K} \text{ and } a \in \mathcal{T}\}$);
- $checksign_X(k^{-1}, [a]_k)$ (*signature verification function* with domain $\{(k^{-1}, [a]_k) \mid k \in \mathcal{K}$ and $a \in \mathcal{T}\}$).

Note that function *checksign* does not produce new terms since its primary task is to verify if its entry term is in its domain. Such a verification function is treated as a function with no output term and the additional condition $a \in \text{dom}(\lambda)$.

The *prefix* of SPPA, ranged over by μ, are obtained as follows:

- $\overline{c}(a)$ (*output prefix*);
- $c(x)$ (*input prefix*);
- $\lambda(a : x)$ (*functional prefix* with $x \notin fv(a)$);
- τ (*silent prefix*);

where a is any term. The set of free variables of a prefix μ, denoted by $fv(\mu)$, is defined as follows: $fv(\overline{c}(a)) = fv(a)$, $fv(c(x)) = \{x\}$, $fv(\lambda(a : x)) = fv(a) \cup \{x\}$ and $fv(\tau) = \emptyset$.

Let μ be a prefix, let a be a term, and let x, y be variables. The *agents* of SPPA, ranged over by P and Q, are constructed as follows:

$$
\begin{array}{lll}
P, Q & ::= & \\
 & \mid \quad \mathbf{0} & (\textit{empty agent}) \\
 & \mid \quad \mu.P & (\textit{prefix agent}) \\
 & \mid \quad P[a/x] & (\textit{assignment}) \\
 & \mid \quad P + Q & (\textit{sum}) \\
 & \mid \quad P|Q & (\textit{parallel composition}) \\
 & \mid \quad P\backslash L & (\textit{restriction}) \\
 & \mid \quad [x = y]\ P & (\textit{match}) \\
 & \mid \quad P/\mathcal{O} & (\mathcal{O}\textit{-observation})
\end{array}
$$

where L is set and $\mathcal{O}$ is a partial mapping (both to be clarified in Section 2.3). From this syntax, recursion is dealt with by using agent names (e.g. by writing $P = \mu_1.\mu_2.P$). Further, we often write $\sum_{1 \le i \le n} P_i$ instead of $P_1 + \ldots + P_n$, and $\prod_{1 \le i \le n} \mu_i.P$ instead of $\mu_1 \ldots \mu_n.P$.

We define $fv(P)$, the set of *free variables* of P, as the set of variables x appearing in P that are not in the scope of an input prefix $c(x)$ or a functional prefix $\lambda(a : x)$. When $x \in fv(P)$, we often write $P(x)$ (with $P(x_1)(x_2) = P(x_1, x_2)$, and so on) and $P(a)$ instead of $P[a/x]$ where every free occurrence of x in P is set to a. Otherwise the variable x is said to be *bound*. A *closed agent*, or simply a *process*, is an agent P such that $fv(P) = \emptyset$. Intuitively, processes and agents are used to specify the principals of a security protocol. In that case, an agent P has a unique identifier $X(P)$, relating it to its principal, such that $X(P) = X(Q)$ for every sub-agent Q.

2.3 Semantics of SPPA

In order to establish an annotation upon the semantics of a SPPA process, we introduce the concept of markers. This purely semantic concept is used to tag public channels and give a description of the current state of the process (or protocol). Thus, markers have specific semantics that restrict their occurrence to particular steps of a process. We assume a finite set Δ of *markers* that may be parametrized by closed terms. We write $\mathrm{dom}(\delta)$ to denote the domain of the marker δ. Furthermore, we assume that every marker belongs to a particular principal and we consider the sets Δ_X, for every principal $X \in \mathcal{I}$, partitioning Δ. For every public channel $c \in C$ and every principal X, we consider two markers $\overline{\delta_{cX}}$ and δ_{cX} such that $\mathrm{dom}(\overline{\delta_{cX}}) = \mathrm{dom}(\delta_{cX}) = \mathrm{dom}(c)$. As an example, we often use a marker $init_X$ to indicate that the principal X has initiated a protocol run.

Let a be a closed term. The *actions* of SPPA are the following:

- $\overline{c}(a)$ (*output action* of sending $a \in \mathrm{dom}(c)$ over the public channel c);
- $c(a)$ (*input action* of receiving $a \in \mathrm{dom}(c)$ over the public channel c);
- $\lambda(a : b)$ (*functional action* of calling private function λ with entry term $a \in \mathrm{dom}(\lambda)$ and producing the closed term $b = \lambda(a)$);
- $\delta(a)$ (the *marker action*, with $a \in \mathrm{dom}(\delta)$);
- τ (the *internal action*).

Furthermore, we also consider *fail actions* $\lambda(a : \mathtt{fail})$ where λ is a non-verification function and $a \notin \mathrm{dom}(\lambda)$. Roughly speaking, an action is either a closed prefix or a marker action. We write Act to denote the set of all actions and we use α to range over this set. We also consider the set Act_X of actions observable only by a principal X defined by:

$$Act_X = \{\lambda(a : b) \in Act \mid \lambda \in \Lambda_X \text{ and } a \in \mathcal{T}\} \cup \{\delta(a) \in Act \mid \delta \in \Delta_X \text{ and } a \in \mathrm{dom}(\delta)\}.$$

We often use C to denote both the set of public channels and the set of output and input actions.

An *observation criterion* is a partial mapping $\mathcal{O} : Act^* \mapsto Act^*$ that intends to express equivalence between process behaviors. Two sequences of actions γ_1 and γ_2 are said to carry out the same observation α whenever $\gamma_1, \gamma_2 \in \mathcal{O}^{-1}(\alpha)$.

Example 1. For any set of actions $L \subseteq Act \setminus \{\tau\}$, we consider the observation criterion $\mathcal{O}_L$ defined as follows:

$$\mathcal{O}_L^{-1}(\alpha) = \begin{cases} (Act \setminus L)^* \ \alpha \ (Act \setminus L)^* & \text{if } \alpha \in L \\ (Act \setminus L)^* & \text{if } \alpha = \tau. \end{cases}$$

Only behaviors from the set L are observable through this observation criterion. In particular, we have a natural observation criterion $\mathcal{O}_{Act_X \cup C}$, often called $\mathcal{O}_X$, describing the actions observable by a principal X.

The operational semantics of a process can be viewed as an extension of the usual notion of a non-deterministic automaton where we generally do not consider final states. Let a be a closed term, let $L \subseteq Act$ and let P, P', Q and Q' be agents. The operational semantics of SPPA is defined as follows:

Output $\dfrac{\overline{\delta_c} \in \Delta_{X(P)}}{\overline{c}(a).P \xrightarrow{\overline{\delta_c}(a)} Q \xrightarrow{\overline{c}(a)} P}$ where Q is a new process name.

Input $\dfrac{a \in \mathrm{dom}(c)}{c(x).P \xrightarrow{c(a)} P[a/x]}$

Marker $\dfrac{P \xrightarrow{c(a)} P' \quad \text{and} \quad \delta_c \in \Delta_{X(P)}}{P' \xrightarrow{\delta_c(a)} P'}$

Function $\dfrac{b{=}\lambda(a) \quad \text{and} \quad \lambda \in \Lambda_{X(P)}}{\lambda(a{:}x).P \xrightarrow{\lambda(a:b)} P[b/x]}$

Fail $\dfrac{a{\notin}\mathrm{dom}(\lambda) \quad \text{and} \quad \lambda \in \Lambda_{X(P)}}{\lambda(a{:}x).P \xrightarrow{\lambda(a:\mathtt{fail})} \mathbf{0}}$

Sum $\dfrac{P \xrightarrow{\alpha} P'}{P{+}Q \xrightarrow{\alpha} P'}$ and $\dfrac{Q \xrightarrow{\alpha} Q'}{P{+}Q \xrightarrow{\alpha} Q'}$

Parallel $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$ and $\dfrac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$

Synchronization $\dfrac{P \xrightarrow{\overline{c}(a)} P' \quad \text{and} \quad Q \xrightarrow{c(a)} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$ and $\dfrac{P \xrightarrow{c(a)} P' \quad \text{and} \quad Q \xrightarrow{\overline{c}(a)} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

Restriction $\dfrac{P \xrightarrow{\alpha} P' \quad \text{and} \quad \alpha \notin L}{P\backslash L \xrightarrow{\alpha} P'\backslash L}$

Match $\dfrac{P \xrightarrow{\alpha} P'}{[a{=}a]\ P \xrightarrow{\alpha} P'}$

$\mathcal{O}$-Observation $\dfrac{P \xrightarrow{\gamma} P' \quad \text{and} \quad \gamma \in \mathcal{O}^{-1}(\alpha)}{P/\mathcal{O} \xrightarrow{\alpha} P'/\mathcal{O}}$.

where the *computation* $P \xrightarrow{\gamma} P'$, for a sequence of actions $\gamma = \alpha_0\alpha_1 \ldots \alpha_n \in Act^*$, stands for the the finite string of transitions satisfying $P \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P'$.

An agent P' is a *derivative* of P if there is a computation $P \xrightarrow{\gamma} P'$ for some $\gamma \in Act^*$. We shall frequently make use of the set $\mathcal{D}(P) = \{P' \mid \exists_{\gamma \in Act^*}\ P \xrightarrow{\gamma} P'\}$, the set of P's derivatives. Note that $X(Q) = X(P)$ for every $Q \in \mathcal{D}(P)$.

2.4 Observation-dependent Bisimulation

For the following, we need Milner's notions of (strong) simulation, noted $\sqsubseteq$, and (strong) bisimulation, noted $\approx$ [18]. The concept of $\mathcal{O}$-bisimulation, called $\mathcal{O}$-congruence by Boudol [6], captures the notion of behavioral indistinguishability through an observation criterion $\mathcal{O}$.

Definition 1. Let $\mathcal{O}$ be an observation criterion.

1. The process P is *$\mathcal{O}$-simulated* by the process Q whenever $P/\mathcal{O} \sqsubseteq Q/\mathcal{O}$. In this case we write $P \sqsubseteq_{\mathcal{O}} Q$.

2. The process P is $\mathcal{O}$-*bisimilar* to the process Q whenever $P/\mathcal{O} \approx Q/\mathcal{O}$, and we write $P \approx_{\mathcal{O}} Q$.

Example 2. Consider the weak criterion $\mathcal{O}_{Vis}$, where $Vis = Act \setminus \{\tau\}$ is the set of visible actions. We can easily see that $\mathcal{O}_{Vis}$-bisimulation corresponds to Milner's weak bisimulation [18].

3 Admissible Interference in Process Algebra

Given a partition of the set Vis of visible actions into three sets Lo, Hi and Dwn holding respectively for the sets of low-level (or public), high-level (or private) and downgrading-level (or De-classifying) observable actions and such that $Lo = \overline{Lo}$, $Hi = \overline{Hi}$ and $Dwn = \overline{Dwn}$. The following formulation of non-interference requires that a process $\mathcal{O}_{Vis}$-simulates its $\mathcal{O}_{Lo}$-observation. Thus, roughly speaking, bisimulation-based strong non-deterministic non-interference (BSNNI) states that any low-level observable behavior has to be also a high-level process behavior, in order to disallow any correlation between a high-level behavior and a low-level observation.

Definition 2. Process P satisfies BSNNI if

$$P/\mathcal{O}_{Lo} \sqsubseteq_{\mathcal{O}_{Vis}} P.$$

It is not difficult to prove, that this property coincides with bisimulation-based strong non-deterministic non-interference as proposed by Focardi and Gorrieri [10].

Admissible interference refers to the information flow properties that require that systems admit information flow from the high-level to the low-level only through specific downgrading channels. To capture this property, it was proposed [20] that any agent P' derived from P and executing no downgrading action be required to satisfy non-interference. More precisely, for P to satisfy intransitive non-interference $P' \setminus Dwn$ must satisfy non-interference for every $P' \in \mathcal{D}(P)$. Rephrasing it in the context of BSNNI as the non-interference property yields the definition of BNAI.

Definition 3. Process P satisfies *bisimulation-based non-deterministic admissible interference* (BNAI) if

$$\forall_{P' \in \mathcal{D}(P)} \quad (P' \setminus Dwn)/\mathcal{O}_{Lo} \ \sqsubseteq_{\mathcal{O}_{Vis}} \ (P' \setminus Dwn).$$

The next theorem presents an algebraic characterization of BNAI based on $\mathcal{O}_{Lo}$-bisimulation.

Theorem 4 (Unwinding Theorem for BNAI). *The process P satisfies BNAI if and only if*

$$\forall_{P' \in \mathcal{D}(P)} \quad P' \setminus Dwn \ \approx_{\mathcal{O}_{Lo}} \ P' \setminus (Dwn \ \cup \ Hi).$$

The proof of Theorem 4 can be found in a previous paper [12].

4 Using Admissible Interference to Detect Potential Denial of Service Attacks

In this section, we use information flow methods to detect possible denial of service attacks on a protocol. To achieve this, we verify whether an intruder may cause interference on the protocol. We are particularly interested in the following two types of denial of service attacks.

Resource exhaustion attacks At any step of the protocol, an intruder sends a fake message in order to waste the victim's resource processing it. In that case we should only consider fake messages that require not much of the intruder's resource and cause a greater waste for the victim.

Authentication attacks An intruder persuades a principal to participate in a fake protocol run. In such attacks, an intruder usually uses the identity of another principal in order to dupe the victim by making him wrongly believe that he received a service (e.g. an authentication or a shared key).

However, we allow any interference coming from an intruder behaving properly. Such honest behaviors include initiating a real protocol run (with its own identifier and no fake message) and replying properly to a protocol run invitation. This assumption of allowing honest intruder's behaviors is often omitted in the literature, but is essential in order to view the intruder as a legitimate user. Admissible interference helps us achieve this goal by allowing an enemy process to cause harmless interference on the protocol through predetermined actions called *admissible attacks*. The set of admissible attacks, noted Γ, is a subset of the set Act_E containing actions that correspond to honest behaviors.

The main contribution of this paper is a validation method for security protocols against resource exhaustion attacks. A similar method for detecting authentication attacks was presented in previous papers [12, 13].

4.1 Specification of Security Protocols

In order to specify a security protocol in SPPA, we use the classic approach [9, 24] of specifying the principals as concurrent processes. Each of these processes has its own identifier and initial knowledge describing the starting information possessed by a principal. The initial knowledge of a principal commonly contains its secret keys, any public key and identifier, fresh nonce, and any other information required by the protocol. However, the initial knowledge of a principal, especially in the case of an intruder, should not contain secret information like the other principals secret keys.

A security protocol P is specified as:

$$P(a_A, a_B, a_1, \ldots, a_n) = (\ A(a_A) \mid B(a_B) \mid S_1(a_1) \mid \ldots \mid S_m(a_m)\ ) \setminus C$$

where C is the set of public channels used by P (commonly, there is one channel for every step of a given protocol run). The process A always specifies the initiator of the protocol and, likewise, the process B always specifies the target of the protocol. The other processes S_i specify principals contributing to the protocol (e.g servers). The closed terms a_A, a_B, $a_1 \ldots, a_m$ are tuples corresponding, respectively, to the initial knowledge of the principals. For the purpose of this paper, we fix a protocol P specified as above and we assume that $A, B, S_1, \ldots, S_m \in \mathcal{I}$.

4.2 Specification of an Enemy Process

Given a security protocol, we are interested in studying its behavior while running in a hostile environment. More precisely, we want to make sure that the protocol acts "correctly" in any given critical situation. Since such hostile environments need to be related to our process algebra, they are specified as enemy processes attempting to attack the protocol through its public channels. We consider a unique enemy identifier $E \in \mathcal{I}$ and sets of enemy functions Λ_E and enemy markers Δ_E. Thus, every enemy process is related to the same enemy identifier, which implies the uniqueness of the observation criterion $\mathcal{O}_E$.

In order to achieve a resource exhaustion attack, an intruder usually needs to initiate several protocol runs, each exploiting the same flaw. For this reason, we only consider attacks where the

intruder is the protocol's initiator. Thus, following the notation established above, the interaction of an enemy process $E(a_E)$ with the protocol P is written as follows:

$$P_E(a_E, a_B, a_1, \ldots a_n) = (\ E(a_E) \mid B(a_B) \mid S_1(a_1) \mid \ldots \mid S_m(a_m)\ ) \setminus C$$

where a_E is a tuple of initial knowledge available to the attacker.

4.3 Specification of Denial of Service as an Information Flow Property

The following approach of assigning cost to actions was inspired by Meadows' cost-based framework [17]. In order to compare resource spending, we consider an ordered set of costs $\langle \mathcal{C}, < \rangle$ and a cost function

$$\rho : Act \mapsto \mathcal{C}$$

where $\rho(\alpha)$ is the quantity of resources, namely the cost, used to execute the action α. Further, we assume that $\rho(\tau) \leq \rho(\alpha)$, for every $\alpha \in Act$. For $n \in \mathcal{C}$, we consider the following sets:

- $Act_X^n \ = \ \rho^{-1}(n) \ \cap \ Act_X$,
- $\Phi_X^n \ = \ \bigcup_{i \leq n} Act_X^i$,
- $\Omega_X^n \ = \ \bigcup_{i \geq n} Act_X^i$, and
- $\Gamma^n \ = \ \Gamma \ \cup \ \Omega_E^n$, where Γ is the set of admissible attacks.

Given a principal X, Act_X^n is the set of actions of cost n observable by X, Φ_X^n is the set of actions of cost lesser or equal to n observable by X, and Ω_X^n is the set of actions of cost greater or equal to n observable by X. For simplicity, we assume that $\mathcal{C} = \{0, 1, \ldots N\}$ with usual linear order.

The main goal of this paper is to establish a model checking method that uses a security property against potential denial of service attacks. For that purpose, first we need to consider every enemy process (definable in SPPA). Given such an enemy process, we proceed to verify for every cost n if the set Φ_E^n of enemy actions of cost n or less causes interference on the set Ω_B^{n+1} of B's actions of cost greater than n. As usual, by interference we mean an action from Φ_E^n causing the occurrence of an action from Ω_B^{n+1} that would have not occurred otherwise. The following definition introduced an information flow property based on BNAI.

Definition 5 (Impassivity). Protocol P is *impassive* if, for every enemy process E,

$$\forall_{n \in \mathcal{C} \setminus \{N\}} \quad (P_E \setminus \Gamma^{n+1}) / \mathcal{O}_{B^{n+1}} \ \sqsubseteq_{\mathcal{O}_{B^{n+1}, E^n}} \ P_E \setminus \Gamma^{n+1}$$

where $\mathcal{O}_{B^{n+1}} = \mathcal{O}_{\Omega_B^{n+1}}$ and $\mathcal{O}_{B^{n+1}, E^n} = \mathcal{O}_{\Omega_B^{n+1} \cup \Phi_E^n}$.

Given an enemy process E, impassivity is satisfied whenever no behavior from E, except admissible attacks, causes interference on more costly behaviors of B. It is important to note that, unlike BNAI, impassivity does not verify whether every derivable process satisfies non-interference. This is due to the fact that the detection of resource exhaustion attacks requires the omission of any attack, namely interference of E on B, that are preceded by a costly enemy action. The following unwinding theorem serves as an bisimulation-based algorithm for impassivity.

Theorem 6. *Protocol P is impassive if and only if, for every enemy process E,*

$$\forall_{n \in \mathcal{C} \setminus \{N\}} \quad P_E \setminus \Gamma^{n+1} \ \approx_{\mathcal{O}_{B^{n+1}}} \ P_E \setminus (\Gamma^{n+1} \ \cup \ \Phi_E^n).$$

Since enemy processes may execute actions from Act_E, Theorem 6 may be improved by directly put the restrictions $\backslash\Gamma^{n+1}$ and $\backslash(\Gamma^{n+1} \cup \Phi_E^n)$ on the enemy process.

Theorem 7. *Protocol P is impassive if and only if, for every enemy process E,*

$$\forall_{n \in \mathcal{C}\backslash\{N\}} \quad Q_n \approx_{\mathcal{O}_{B^{n+1}}} Q'_n$$

where $Q_n = (\ E \backslash \Gamma^{n+1} \mid B \mid S_1 \mid \ldots \mid S_m\ )$ *and* $Q'_n = (\ E \backslash (\Gamma^{n+1} \cup \Phi_E^n) \mid B \mid S_1 \mid \ldots \mid S_m\ )$.

Remark 8. The impassivity property, as well as its unwinding theorem, could be modified to accomodate the following variants.

1. Given a principal $S \in \mathcal{I}$ participating to the protocol P, we say that P is *impassive with respect to S* whenever P satisfies the property, or equivalently the unwinding theorem, obtained from impassivity by replacing each identifier B by S. This property is useful to detect denial of service attacks against a server S acting as a principal in a security protocol.

2. We can restrict the impassivity property by only detecting, for every $n \in \mathcal{C}$, inadmissible interference coming from enemy actions of cost n on B's actions of cost $n+2$ or higher. Such alteration of our property is often useful since the scale for establishing the actions costs is mostly case dependant. This property allows protocols to use increasing authentication mechanisms.

Both our definition and unwinding theorem of impassivity suffer from a universal quantification over enemy processes. This problem can be by-passed by defining a strongest enemy process and verifying our property only with this enemy process. Such a strongest attacker has the capability to $\mathcal{O}_{Vis}$-simulate any other enemy process. It is built in section 4.4.

4.4 The Strongest Attacker

Given a cryptographic protocol P specified in our process algebra, we are interested in defining the most powerful enemy attempting to attack P. Focardi [9] proposed a notion of most powerful enemy with respect to trace pre-order. In this paper, we present a more general result based on weak simulation. Thus, our most powerful enemy shall be able to simulate any other enemy process definable in our process algebra. More specifically, we are mainly interested in the behaviors of the protocol interacting with different enemy processes.

Definition 9.
1. Given two enemy processes E and E', we say that E is *stronger* (*with respect to P*) than E' whenever $P_{E'} \sqsubseteq_{\mathcal{O}_{Vis}} P_E$. In that case, we write $E' \leq_P E$.

2. The *strongest attacker* for the protocol P is an enemy process $\mathcal{E}$ such that for every enemy process E, we have $E \leq_P \mathcal{E}$.

Lemma 10. *If $E' \sqsubseteq_{\mathcal{O}_E} E$, then $E' \leq_P E$.*

Proof. First we see that an enemy process E may only execute actions from $Act_E \cup C$. This is a direct consequence of the semantic rules `Output`, `Marker` and `Function`. Thus we have $E/\mathcal{O}_E \approx E/\mathcal{O}_{Vis}$. From this fact, we obtain that

$$E' \sqsubseteq_{\mathcal{O}_E} E \quad \Longleftrightarrow \quad E' \sqsubseteq_{\mathcal{O}_{Vis}} E$$

for any enemy processes E' and E. The conclusion then follows from the fact (proved in [18]) that weak simulation is a congruence with respect to the concurrent and restriction operators. □

From this lemma, we see that in order to find a strongest attacker for a protocol P, it is enough to find an enemy process $\mathcal{E}$ that may $\mathcal{O}_E$-simulate any other enemy process. Note that the converse of Lemma 10 is not true in general.

4.4.1 A Generic Enemy Process

In this section, we construct a candidate for the role of strongest attacker. The *generic enemy process* for the protocol P is the process $\mathcal{E}(m_E)$ where $\mathcal{E}(x_1, \ldots, x_n)$ (for any n) is defined as follows:

$$\begin{aligned}\mathcal{E}(x_1, \ldots, x_n) = & \sum_{1 < i \leq n} \tau.\, \mathcal{E}(x_i, \ldots, x_1, \ldots, x_n) \\ & + \sum_{c \in C} c(y).\, \mathcal{E}(x_1, \ldots, x_n, y) \\ & + \sum_{c \in C} \overline{c}(x_1).\, \mathcal{E}(x_1, \ldots, x_n) \\ & + \sum_{\lambda \in \Lambda_E} \lambda(x_1, \ldots, x_i : y).\, \mathcal{E}(x_1, \ldots, x_n, y)\end{aligned}$$

Lemma 11. *Let $E(a_E)$ be an enemy process and let b be a closed term. If b is obtainable from $E(a_E)$, then b is also obtainable from the generic enemy process $\mathcal{E}(m_E)$.*

Proof. We proceed by induction on the length of the derivation used by $E(a_E)$ to obtain b. The case where $E(a_E)$ initially contains b, as a component of a_E, is straightforward since m_E is assumed to be maximal, thus also contains b.

Now assume that the statement of the lemma is true for any closed term obtainable within n transitions. Consider b obtainable from $E(a_E)$ within $n+1$ transitions. In that case, we have processes $Q'(b'), Q''(b'') \in \mathcal{D}(E(a_E))$ such that $Q'(b')$ is derivable from $E(a_E)$ in less than $n+1$ transitions, $Q'(b') \xrightarrow{\alpha} Q''(b'')$, and b'' contains b. By the induction hypothesis, b' is obtainable from $\mathcal{E}(m_E)$. Consider such a process $\mathcal{E}(a') \in \mathcal{D}(\mathcal{E}(m_E))$ with a' containing b'. By the definition of $\mathcal{E}(x_1, \ldots, x_n)$ and since we must have $\alpha \in Act_E \cup C \cup \{\tau\}$, there is a finite m such that $\tau^m \alpha$ is a computation of the process $\mathcal{E}(a')$. Thus we have

$$\mathcal{E}(a') \xrightarrow{\tau} \mathcal{E}(a'_1) \xrightarrow{\tau} \ldots \xrightarrow{\tau} \mathcal{E}(a'_m) \xrightarrow{\alpha} \mathcal{E}(a'')$$

and any new closed term obtained by α in $Q'(b') \xrightarrow{\alpha} Q''(b'')$ will also be gained in $\mathcal{E}(a'_m) \xrightarrow{\alpha} \mathcal{E}(a'')$ since a'_m contains b' (because process $\mathcal{E}$ never trough away terms). Hence a'' contains b''. We may then conclude that b'' is obtainable from $\mathcal{E}(a')$, thus from $\mathcal{E}(m_E)$. □

Theorem 12. *The process $\mathcal{E}(m_E)$ is the strongest attacker for the protocol P.*

Proof. Let $E(a_E)$ be any enemy process definable in our process algebra. From Lemma 10, it is enough to show that

$$E(a_E) \;\sqsubseteq_{\mathcal{O}_E}\; \mathcal{E}(m_E).$$

By Lemma 11, in order to establish this $\mathcal{O}_E$-simulation, it is enough to see that for any $Q(b) \in \mathcal{D}(E(a_E))$, we have

$$Q(b) \;\sqsubseteq_{\mathcal{O}_E}\; \mathcal{E}(b).$$

This fact is true since whenever $Q(b) \xrightarrow{\alpha} Q'(b')$, then we must have $\alpha \in Act_E \cup C \cup \{\tau\}$ and there is a finite m such that $\tau^m \alpha$ is a computation of $\mathcal{E}(b)$. Thus we have $Q'(b) \;\sqsubseteq_{\mathcal{O}_E}\; \mathcal{E}(b)$. □

An example of this strongest attacker is given in [13] for the Needham-Schroeder authentication protocol.

5 Application to the 1KP Secure Electronic Payment Protocol

The family of protocols iKP ($i = 1, 2, 3$) for secure electronic payments over the Internet was developed by a group from IBM Research Division [4, 3]. The protocols implement credit card based transactions between a buyer and a seller by using the existing financial network for clearing and authorization. All three iKP protocols follow the same steps and only differ by the content of the message sent at each step. The iKP protocols use public-key encryption, strong collision-resistant one-way hash function and public-key signature. The public and secret keys of a principal X are noted by pk_X and sk_X (with $pk_X^{-1} = sk_X$). For the purpose of this paper, we shall only study 1KP, the simplest of the trio, which proceeds as follows:

$$\begin{array}{lcc}
\text{Message 1:} & A & \xrightarrow{salt_A, h(r_A, ban_A)} & B \\
\text{Message 2:} & B & \xrightarrow{clear} & A \\
\text{Message 3:} & A & \xrightarrow{\{slip\}_{pk_S}} & B \\
\text{Message 4:} & B & \xrightarrow{clear, h(salt_A, desc), \{slip\}_{pk_S}} & S \\
\text{Message 5:} & S & \xrightarrow{resp, [resp, h(common)]_{sk_S}} & B \\
\text{Message 6:} & B & \xrightarrow{resp, [resp, h(common)]_{sk_S}} & A
\end{array}$$

where $common = (price, B, tid_B, date, n_B, h(r_A, ban_A), h(salt_A, desc))$, $clear = (B, tid_B, date, n_B, h(common))$ and $slip = (price, h(common), ban_A, r_A, exp_A)$. First, the buyer A sends to the seller B a random number $salt_A$ and a second random number r_A hashed with the buyer's account number ban_A (e.g., credit card number). Secondly, the seller B replies with his identifier, the transaction identifier tid_B, the date (or timestamp) $date$, a fresh nonce n_B, and the hashed value of the field $common$, where $price$ and $desc$ are the amount and the description of the purchase. Thirdly, the buyer A sends to B the field $slip$ encrypted with the acquirer's public key pk_S, where exp_A is the expiration date associated to ban_A. The seller B may now request an authorization clearance from the acquirer S for the payment by forwarding the message just received from A along with the field $clear$ and $h(salt_A, desc)$. The acquirer S then proceeds as follows:

1. S makes sure that the values from $clear$ were not used in a previous request;
2. S decrypts $\{slip\}_{pk_S}$ and obtains $slip$;
3. S checks whether the $h(common)$ from $clear$ matches the one from $slip$;
4. S re-constructs $common$, computes $h(common)$ and checks whether it matches the one from $clear$;
5. S sends ban_A and exp_A to the credit card organisation's clearing and authorization system in order to obtain an on-line authorization for the payment.

Once this authorization procedure is finished, the acquirer S sends to the seller B the response $resp$ received from the authorization system along with $h(common)$ and $resp$ signed together with S's secret key. Finally, B checks the acquirer's signature and forwards the previous message to A.

5.1 Specification of 1KP

Every principal of the 1KP protocol is specified separately as follows:

$$
\begin{aligned}
A(a_A) = \;& hash_A(r_A, ban_A : x_1).\ pair_A(salt_A, x_1 : x_2).\ \overline{c_1}(x_2).\\
& c_2(x_3).\ \prod_{1\le i\le 5} extract_A(i, x_3 : x_{3i}).\ checkdate_A(x_{33}).\ hash_A(salt_A, desc : x_4).\\
& pair_A(price, x_{31}, x_{32}, x_{33}, x_{34}, x_1, x_4 : x_5).\ hash_A(x_5 : x_6).\\
& [x_6 = x_{35}]\ pair_A(price, x_5, ban_A, r_A, exp_A : x_7).\ enc_A(pk_S, x_7 : x_8).\ \overline{c_3}(x_8).\\
& c_6(x_9).\ \prod_{i=1,2} extract_A(i, x_9 : x_{9i}).\ checksign_A(pk_S, x_{91}, x_{92}).\ \mathbf{0}\\
B(a_B) = \;& c_1(y_1).\ \prod_{i=1,2} extract_B(i, y_1 : y_{1i}).\ hash_B(y_{11}, desc : y_2)\\
& pair_B(price, B, tid_B, date, n_B, y_2 : y_3).\ hash_B(y_3 : y_4)\\
& pair_B(B, tid_B, date, n_B, y_4 : y_5).\ \overline{c_2}(y_5).\\
& c_3(y_6).\ pair_B(y_5, y_2, y_6 : y_7).\ \overline{c_4}(y_7).\\
& c_5(y_8).\ \prod_{i=1,2} extract_B(i, y_8 : y_{8i}).\\
& checksign_B(pk_S, y_{81}, y_{82}).\ \overline{c_6}(y_8).\ \mathbf{0}\\
\\
S(a_S) = \;& c_4(z_1).\ \prod_{1\le i\le 3} extract_S(i, z_1 : z_{1i}).\ \prod_{1\le i\le 5} extract_S(i, z_{11} : z_{11i}).\\
& checkreplay_S(z_{111}, z_{112}, z_{113}, z_{114}).\ dec_S(sk_S, z_{13} : z_2).\\
& \prod_{1\le i\le 5} extract_S(i, z_2 : z_{2i}).\ [z_{115} = z_{22}]\ hash_S(z_{24}, z_{23} : z_3).\\
& pair_S(z_{21}, z_{111}, z_{112}, z_{113}, z_{114}, z_3, z_{12} : z_4).\ hash_S(z_4 : z_5).\\
& [z_5 = z_{22}]\ clearing_S(z_{23}, z_{25}, z_{21} : z_6).\ pair_S(z_6, z_5 : z_8).\ sign_S(sk_S, z_7 : z_8).\\
& pair_S(z_6, z_8 : z_9).\ \overline{c_5}(y_9).\ S
\end{aligned}
$$

with the respective initial knowledge tuples $a_A = (r_A, salt_A, ban_A, exp_A, desc, price, pk_S)$, $a_B = (desc, price, pk_S)$ and $a_S = (pk_S, sk_S)$. From this specification, we may assume that

$$\Lambda_X = \{extract_X, pair_X, hash_X, enc_X, dec_X, sign_X, checksign_X, checkdate_X\}$$

for any principal $X \neq S$. Note that 1KP requires two hash functions, one with a single argument and a second with two arguments. Note that our specification uses a unique function name. The verification function $checkdate_X(date)$ validates $date$ within a pre-defined time skew, but since our model does not deal with time, we shall assume that its domain is composed of pre-determined values. The set Λ_S of the acquirer's private functions contains, in addition to every functions above, functions $checkreplay_S$ and $clearing_S$. The verification function $checkreplay_S(X, tid, date, n)$ is intended to make sure that there were no previous request with same terms, thus its domain depends on the current stored terms of the acquirer process. The function $clearing_S(ban, exp, price : resp)$ validates the terms $ban, exp, price$ in order to get an authorization for the payment.

For this specification of the 1KP protocol, we consider the following markers: $init_X = \overline{delta_{c_1X}}$, $request_X = \underline{delta_{c_1X}}$, $commit_X = \overline{delta_{c_3X}}$, $authinit_X = \overline{delta_{c_4X}}$ $authrequest_X = delta_{c_4X}$, $authresp_X = \overline{delta_{c_5X}}$ and $confirm_X = \overline{delta_{c_6X}}$.

Assigning costs to actions is always problematic since it depends on many local factors like the complexity of the encryption scheme, the hash function and the signature scheme. For the purpose of this paper, we assume that signature and signature verification are the most costly operations. Then comes encryption, decryption and hashing with a medium cost, as well as any clearing action done by the acquirer. The other actions, including all the marker actions, are assumed to be of equal low cost. Thus, we consider the set of costs $\mathcal{C} = \{1, 2, 3\}$ and the cost function ρ defined as follows:

$$\rho^{-1}(1) = \bigcup_{X\in\mathcal{I}} \{extract_X(i, a : \pi_i(a)), pair_X(a_1, \ldots, a_n : (a_1, \ldots, a_n)), checkdate_X(a)\} \cup \{checkreplay_S(a)\} \cup \{\delta(a) \mid \delta \in \Delta\};$$

$$\rho^{-1}(2) = \bigcup_{X\in\mathcal{I}} \{enc_X(k, a : \{a\}_k), dec_X(k^{-1}, \{a\}_k : a), hash_X(a : h(a))\} \cup \{clearing_S(a_1, a_2, a_3 : b)\};$$

$$\rho^{-1}(3) = \bigcup_{X\in\mathcal{I}} \{sign_X(k, a : [a]_k), checksign_X(k, [a]_k)\}.$$

5.2 Denial of Service Attacks on 1KP

Consider the following enemy process:

$$E(m_1, m_2, m_3) \;=\; pair_E(m_1, m_2 : x_1).\ \overline{c_1}(x_1).\ c_2(x_2).\ \overline{c_3}(m_3).\ \mathbf{0}$$

where m_1, m_2, m_3 are arbitrary values. With this enemy process, we see that the 1KP protocol is not impassive (in terms of Definition 5) since

$$P_E \setminus \Gamma^2 \;\not\approx_{\mathcal{O}_{B^2}}\; P_E \setminus (\Gamma^2 \cup \Phi_E^1).$$

Our enemy process may pursue a denial of service attack on the 1KP protocol from only three freshly generated messages m_1, m_2, m_3 as follows: first, E sends any two fake messages m_1 and m_2 to B, which, in response, executes two hashing actions. This is detected by the fact that the marker action $init_E(m_1, m_2)$ of cost 1 causes interference on the actions $hash_B(m_1, desc : h(m_1, desc))$ and $hash_B((price, B, tid_B, date, n_B, h(m_1, desc)) : h(\ldots))$ of cost 2. Furthermore, we see that our enemy process E may also pursue an denial of service attack on the acquirer process. This attack happens after B replies to E. Then E sends back a fake message m_3 in order to initiate the authorization procedure. After receiving m_3, B forwards it to S. This leads S to execute a decryption action of cost 2 (which fails since m_3 in not encrypted). In that case we have

$$P_E \setminus \Gamma^2 \;\not\approx_{\mathcal{O}_{S^2}}\; P_E \setminus (\Gamma^2 \cup \Phi_E^1)$$

since the action $commit_E(m_3)$ causes interference on the action $dec_S(sk_S, m_3 : \texttt{fail})$. This denial of service attack on S may also be launched by the following smaller enemy process:

$$E'(m_1, m_2, m_3) \;=\; pair_E(m_1, m_2, m_3 : x_1).\ \overline{c_4}(x_1).\ \mathbf{0}$$

where m_1 is a tuple faking $clear$, m_2 is a value faking $h(salt_A, desc)$ and m_3 is a value faking $\{slip\}_{pk_S}$.

Another denial of service attack on the acquirer S may be launched by the following enemy process:

$$E''(a_E) = \; hash_E(common' : x_1).\ pair_E(price', x_1, ban', r', exp' : x_2).\ enc_E(pk_S, x_2 : x_3).\ pair_E(B', tid', date, n', x_1 : x_4).\ pair_E(x_4, m_1, x_3 : x_5).\ \overline{c_4}(x_5).\ \mathbf{0}$$

where a_E contains a tuple $common' = (price', B', tid', date, m_1, m_2)$ faking $common$, and values ban', r' and exp' faking A's initial knowledge. The enemy process E'' may force S to execute a clearing action, which yields a negative response, followed by a signature action. Those two actions that we assumed of cost 3 come at the price of an intruder hash action and encryption action, both of cost 2.

6 Future and Related Works

Toward a CAD tool to detect potential DoS attacks upon security protocols We are designing and implementing at *École Polytechnique de Montréal*, a tool to check whether a process satisfies admissible interference or not. An on-line demonstration applet of the tool can be found at `www.crac.polymtl.ca`. We plan to extend this tool to a security protocol compiler. Protocols will be specified using a notation *à la Alice and Bob*, compiled into SPPA processes and then analysed along the lines described in this paper.

Distributed Denial of Service The most lethal distributed denial of service (DDoS) attacks have also cause their share of mayhem. Some tools devoted to DDoS [7, 8, 22] were developped for the analysis of such attacks based on specific malicious applications like *Trin00*, *TFN2K* and *Stacheldraht*. Literature does not seem to contain any formal methods for detecting DDoS, or any other denial of service attacks, except for Meadows' cost-based framework [17]. We strongly believe that the method presented in this paper is suitable for detecting potential DDoS attacks on security protocols.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[2] R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. Technical Report 3915, INRIA, March 2000.

[3] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, and M. Waidner. Design, implementation and deployment of the *i*KP secure electronic payment system. *IEEE Journal of Selected Areas in Communications*, 18(4), April 2000.

[4] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. ikp – a family of secure electronic payment protocols. In *First USENIX Workshop on Electronic Commerce*, May 1995.

[5] C. Bodei, P. Degano, F. Nielson, and H. Nielson. Static analysis for the π- calculus with applications to security. *Information and Computation*, (to appear) 2001.

[6] G. Boudol. Notes on algebraic calculi of processes. In *Logic and Models of Concurrent Systems*, NATO ASI Series F-13, pages 261–303. Springer, 1985.

[7] P. J. Criscuolo. Distributed denial of service trin00, tribe flood network, tribe flood network 2000, and stacheldraht. Technical Report CIAC-2319, Lawrence Livermore National Laboratory, February 2000.

[8] S. Dietrich, N. Long, and D. Dittrich. Analyzing distributed denial of service tools: The shaft case. In *Proceedings of USENIX LISA 2000*, 2000.

[9] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In H. Orman and C. Meadows, editors, *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers University, September 1997. DIMACS Center.

[10] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *J. of Computer Security*, 3(1):5–33, 1994/1995.

[11] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Dependable Computing for Critical Applications*, volume 5, pages 79–100. IEEE Computer Society, 1998.

[12] S. Lafrance and J. Mullins. Bisimulation-based non-deterministic admissible interference and its application to the analysis of cryptographic protocols. *ENTCS*, 61, 2002.

[13] S. Lafrance and J. Mullins. A generic enemy process for the analysis of cryptoprotocols, 2002. Submitted for publication.

[14] L. Lamport. Times, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558– 565, July 1978.

[15] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In *Proc. of TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, 1996.

[16] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[17] C. Meadows. A cost-based framework for analysis of denial of service networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.

[18] R. Milner. *Communication and concurrency*. Prentice-Hall, 1989.

[19] D. Moore, G. Voelker, and S. Savage. Inferring internet denial of service activity. In *USENIX Security Symposium*, August 2001.

[20] J. Mullins. Nondeterministic admissible interference. *Journal of Universal Computer Science*, 6(11):1054–1070, 2000.

[21] J. Mullins and M. Yeddes. Two proof methods for bisimulation-based non-deterministic admissible interference. *Journal of Applied Systems Studies*, 2001. Submitted for publication.

[22] V. Paxson. An analysis of using reflectors in distributed denial-of-service attacks, 2001.

[23] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *14th IEEE Computer Security Foundations Workshop*, pages 174–190, 2001.

[24] S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

[25] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society, IEEE Computer Society Press, May 1997.

École Polytechnique de Montréal
C.P. 6079, Succ. Centre-ville
Montréal (Québec)
H3C 3A7