Modular Preservation of Safety Properties by Cookie-Based DoS-Protection Wrappers

Rohit Chadha, Carl A. Gunter, Jose Meseguer,
Ravinder Shankesi and Mahesh Viswanathan

Dept. of Computer Science, University of Illinois at Urbana-Champaign

Abstract. Current research on verifying security properties of communication protocols has focused on proving integrity and confidentiality using models that include a strong Man-in-the-Middle (MitM) threat. By contrast, protection measures against Denial-of-Service (DoS) must assume a weaker model in which an adversary has only limited ability to interfere with network communications. In this paper we demonstrate a modular reasoning framework in which a protocol $\mathcal{P}$ that satisfies certain security properties can be assured to retain these properties after it is "wrapped" in a protocol $\mathcal{W}[\mathcal{P}]$ that adds DoS protection. This modular wrapping is based on the "onion skin" model of actor reflection. In particular, we show how a common DoS protection mechanism based on cookies can be applied to a protocol while provably preserving safety properties (including confidentiality and integrity) that it was shown to have in a MitM threat model.

1 Introduction

System security has many aspects, including secrecy, authentication, access control, availability, and many target sub-systems such as hardware, network protocols, operating system, application software, and so on. Security properties are typically verified of sub-systems, for example, by showing that a network protocol correctly uses cryptography to preserve a secret. But what we care about is the *end-to-end* security of the whole system, noting that it is a complex combination of its sub-systems. Modularity constructs and modular reasoning about properties are crucial to obtaining such end-to-end security guarantees. In this paper we present some modularity techniques and preservation results of this nature about two quite different kinds of sub-systems and properties, namely, an underlying communication protocol that may enjoy some *safety properties*, either related to security or to broader requirements, and a sub-system protecting against Denial of Service (DoS), which ensures some *availability properties*.

In actual practice, DoS protection mechanisms are not described and implemented in a modular way: typically an underlying protocol is changed in an ad-hoc way. For example, the TCP protocol can be defended against SYN attacks using SYN cookies [8], but requirements such as the need to assure compatibility with clients running diverse versions of TCP prevent a modular addition of this strategy. In particular, the TCP solution cannot be applied directly to other protocols, because it is inter-woven in a non-modular way with the specifics of TCP. Our first contribution in this paper is to consider a common DoS protection mechanism based on cookies, provided and

described in a modular way as a generic "wrapper" that can be applied to an underlying protocol under minimal assumptions. In this way, a DoS protection mechanism becomes highly reusable and modular: one can develop its generic DoS wrapper once and for all. The wrapper then provides the desired DoS protection regardless of the underlying protocol it is applied to (under minimal assumptions): no changes to the underlying protocol are required. Specifically, we leverage ideas from distributed object reflection, where a distributed object can be wrapped by a "meta-object" that mediates its communication, with no required changes to the code of the underlying object. In our treatment we use a simplified version of the "onion skin" model of actor reflection [2], formalized as rewrite theories [13, 32]. We specify in rewriting logic and study in detail a generic wrapper for cookies like the strategy used in TCP SYN cookies. To illustrate this modularity, we sketch how it can be applied to a protocol like Internet Key Exchange (IKE) as was done in IKEv2. Although, our modular approach is applied to cookie mechanism in this paper, we believe that our techniques will apply to other DoS protection mechanisms such as those described in [20, 25].

Our second contribution is to prove that the system composition obtained by adding a cookie-based protection wrapper to a protocol preserves all the safety properties enjoyed by the original protocol. That is, the new availability properties enjoyed by the wrapped protocol are obtained *without losing* any of the safety properties. We obtain this result by specifying the given protocol, the wrapper, and the wrapped protocol as rewrite theories, and proving that: (1) a suitable stuttering simulation exists between the wrapped protocol and the original one; and (2) such simulations preserve all safety properties satisfied by the original protocol. The stuttering simulation constructed as a homomorphic map between the rewrite theories of the protocol with the cookie-wrapper and the original protocol essentially "forgets" the cookies used for DoS protection. We point out here that there is no simulation of the original protocol by the wrapped protocol. This is because the presence of the DoS protection necessarily implies that "malicious" service requests from illegitimate clients, which would be serviced in the absence of the DoS protection, must be ignored by the server . The dropping of the malicious service requests by the server wrapper also implies that we have to assume that the protocols are executed in a lossy environment, which allows the underlying protocol to simulate the dropping of the service requests by loss of messages.

Note that the availability properties enjoyed by the DoS protection mechanism (in this case the cookie mechanism) are, by definition, those of the wrapped protocol. That is, they are *emergent* properties of the corresponding wrapped composition, which typically did not exist in the original protocol. In this paper, we *assume that these availability properties are analyzed separately* by existing methods [30, 28, 1, 4]. The main result of our paper then ensures that: (i) if the underlying protocol, say $\mathcal{P}$, satisfies a set Γ of safety properties; and (ii) if the application $\mathcal{W}[\mathcal{P}]$ of the cookie wrapper $\mathcal{W}$ to protocol $\mathcal{P}$ satisfies a set Δ of availability properties, *then* $\mathcal{W}[\mathcal{P}]$ satisfies *both* Δ and the safety properties Γ.

In the case of security-related safety properties Γ, such as secrecy and authentication properties, enjoyed by a protocol $\mathcal{P}$, such properties are not just enjoyed by $\mathcal{P}$ itself: they are enjoyed by $\mathcal{P}$ *in the context* of a malicious environment that includes a Man-in-the-Middle (MitM) threat, which we can specify with a separate rewrite theory

$\mathcal{I}$. That is, the security-related safety properties Γ are satisfied by the *union* of theories $\mathcal{P} \cup \mathcal{I}$. To cover also these security-related safety properties, we prove a second version of our main theorem corresponding to safety properties that involve a MitM threat, against which the original protocol had been proved secure. We show that such an attacker cannot violate any of the already-proved safety properties for the cookie-wrapped extension of the protocol. Since the assumptions on the underlying protocol are really minimal, our result applies to the preservation of security-related safety properties for a wide range of cryptographic protocols. Therefore, proofs of such properties do not have to be redone after such protocols are subsequently hardened against DoS attacks by a cookie mechanism. Furthermore, since the MitM attacker in the paper is parametrized by an equational theory, our preservation result also applies to safety properties proved in the presence of a MitM attacker that can exploit algebraic properties of the cryptographic constructs used in protocol messages. We discuss one example application of our main result to a concrete protocol, namely, IKEv2 and its cookie mechanism.

The main technical challenge in proving the existence of the stuttering simulation in presence of the MitM threat is that the simulation can no longer be a homomorphic map that just "forgets" cookies. This is because the MitM attacker can intercept the cookies and generate new messages such as embedding the intercepted cookies within encrypted messages. Since the wrappers only perform a check of the accompanying cookies, these encrypted messages may lead to protocol executions which will not be captured by a simulation that forgets the cookies. This issue is resolved by exploiting the capability of the MitM attacker to generate cookies – cookie generation by the wrappers is simulated by the MitM attacker generating and storing new cookies.

The paper is organized into seven sections. Section 2 describes some preliminary concepts. Section 3 describes the underlying protocol semantics. Section 4 describes the stuttering simulation and preservation results for the cookie wrapper. Section 5 describes the MitM intruder model and how safety properties are preserved for the application of the wrapper. Section 6 discusses future work and Section 7 gives conclusions and sketches related work.

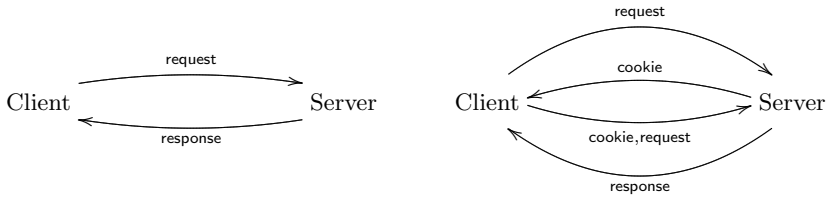
2 Preliminaries

The aim of this paper is to formalize, in a modular way, a simple cookie wrapper and to show that its use in transforming an underlying protocol does not invalidate the safety properties the underlying protocol enjoyed in a MitM model. The semantics of the underlying protocol and the protocol with the cookie wrapper are described using Kripke structures. A next-free safety fragment [36] of the logic LTL is used to specify safety properties. The preservation of safety properties is shown by exhibiting a stuttering simulation between the wrapped protocol and the underlying protocol. The Kripke structures are generated using rewrite theories.

2.1 Cookie-Based DoS Protection

A cookie is used in many network protocols (such as HTTP [16]), by one communicating party A to store some persistent information at B. A cookie k, when presented by

B to A, gives a weak guarantee that B has communicated with A at least once before to get the cookie. This guarantee is weak, because the cookie could be eavesdropped by a MitM intruder, who could impersonate B. However, in a typical (Distributed) Denial of Service flooding attack, the attacker tries to overwhelm a server's limited resources (such as memory or processing power) by sending, with little overhead, a large number of requests using different faked source IP address (see for e.g., [35]). If the cookie mechanism is used the attacker will not receive the cookies sent to the fake IP addresses unless it has a MitM ability. An adversary with MitM ability has already won, so DoS defense mechanisms target to protect against weaker adversaries.



For instance, a simple request-response protocol can be made DoS-resistant by using cookies and the communication pattern shown in the right of the above figure. The basic protocol is modified so that, when first contacted, the server sends a cookie to be stored at the client's side. The server simultaneously retains the ability to retrieve the cookie-value corresponding to that client. From then on, every client request is accompanied by the cookie, that the server can validate before committing any resources to that request. The server can reject all requests without the right cookies, with little overhead (see for e.g., [8]). Since the attacker uses spoofed addresses, it will not receive the cookie response from the server and thus cannot make any resource-intensive requests, thereby preventing the attack.

2.2 Kripke Structures, Safety Properties and Stuttering Simulation

The semantics of the underlying and wrapped protocols are defined using Kripke structures. Given a set of propositions AP, an AP-Kripke structure $\mathcal{A}$ is a tuple $(\mathsf{A}, \rightarrow_{\mathcal{A}}, \mathsf{L}_{\mathcal{A}})$ where the set A is the set of configurations of the protocol, the transition relation $(\rightarrow_{\mathcal{A}} \subseteq \mathsf{A} \times \mathsf{A})$ describes the temporal evolution of the configurations, and the labeling function $\mathsf{L}_{\mathcal{A}} : \mathsf{A} \rightarrow 2^{\mathsf{AP}}$ describes the set of propositions true in a configuration.

The safety-properties that we shall consider in this paper are expressed in the following next-free safety fragment [36] of the logic $\mathsf{LTL}(\mathsf{AP})$ (henceforth called $\mathsf{Safety}\backslash\bigcirc$). The syntax of the fragment $\mathsf{Safety}\backslash\bigcirc$ in BNF notation is:

$$\psi = (p) \;[]\; (\neg p) \;[]\; (\psi \vee \psi) \;[]\; (\psi \wedge \psi) \;[]\; (\psi \,\mathcal{W}\, \psi) \;[]\; (\Box\, \psi)$$

where $p \in \mathsf{AP}$. Here, the modalities $\mathcal{W}$ and $\Box$ are the usual weak-until and always modalities of LTL. We refer the reader to [36] for a formal definition. Please note that we could have also used other characterizations of safety properties such as the syntactic characterization using past operators [27] or the semantic characterization of safety [5].

As discussed above, we shall show that the safety properties are preserved by a wrapped protocol by exhibiting a stuttering simulation between the wrapped protocol and the underlying protocol. Intuitively, a system $\mathcal{A}$ is simulated by a system $\mathcal{B}$ if every execution step of $\mathcal{A}$ can be matched by an execution step of $\mathcal{B}$. Since we are primarily interested in a next-free fragment of LTL, we require that an execution step of $\mathcal{A}$ is matched by zero or more execution steps of $\mathcal{B}$. Formally,

Definition: Given two AP-Kripke structures $\mathcal{A} = (\mathsf{A}, \rightarrow_{\mathsf{A}}, \mathsf{L}_{\mathsf{A}})$ and $\mathcal{B} = (\mathsf{B}, \rightarrow_{\mathsf{B}}, \mathsf{L}_{\mathsf{B}})$ a relation $\mathcal{H} \subseteq \mathsf{A} \times \mathsf{B}$ is said to be a *stuttering simulation* if for all a, b such that $a\mathcal{H}b$, the following hold: $\mathsf{L}_{\mathsf{A}}(a) = \mathsf{L}_{\mathsf{B}}(b)$, and if $a \rightarrow_{\mathcal{A}} a'$ then there exists a natural number $0 \leq j$ and elements $b_0, b_1, \ldots, b_j$ such that

1. $b_0 = b$
2. $b_l \rightarrow_{\mathcal{B}} b_{l+1}$ for all $0 \leq l < j$, and
3. there is a $0 \leq k \leq j$ such that $a\mathcal{H}b_l$ for all $0 \leq l \leq k$ and $a'\mathcal{H}b_l$ for all $k < l \leq j$.

The Kripke structure $\mathcal{A}$ is said to be stuttering simulated by $\mathcal{B}$ if there exists a stuttering simulation $\mathcal{H} \subseteq \mathsf{A} \times \mathsf{B}$.

Stuttering simulations reflect satisfaction of Safety$\backslash\bigcirc$ properties.

Proposition 1 *Given two* AP*-Kripke structures* $\mathcal{A} = (\mathsf{A}, \rightarrow_{\mathsf{A}}, \mathsf{L}_{\mathsf{A}})$ *and* $\mathcal{B} = (\mathsf{B}, \rightarrow_{\mathsf{B}}, \mathsf{L}_{\mathsf{B}})$ *, a stuttering simulation* $\mathcal{H} \subseteq \mathsf{A} \times \mathsf{B}$*, configurations* a, b *such that* $a\mathcal{H}b$ *and a next-free safety formula,* $\psi \in$ Safety$\backslash\bigcirc$*, we have* $\mathcal{B}, b \models_{\mathcal{B}} \psi \Rightarrow \mathcal{A}, a \models_{\mathcal{A}} \psi$*.*

2.3 Rewriting Logic

We specify the configurations of a protocol as the algebraic data type associated to an *order-sorted equational theory* (Σ, E) [17], where the signature Σ specifies the sorts, a subsort relation interpreted as subset inclusion in the algebras, and the constants and function symbols, and where E is a set of Σ-equations. The algebraic data type associated to (Σ, E) is the *initial algebra* $T_{\Sigma/E}$ [17]. In our protocols, the signature Σ will contain a sort Conf of object and message configurations as the chosen sort of configurations, so that the set of protocol configurations is the set $T_{\Sigma/E,\mathsf{Conf}}$. The theory (Σ, E) only specifies the *statics* of a protocol. A protocol $\mathcal{P}$, including its *dynamics*, is specified as a *rewrite theory* $\mathcal{P} = (\Sigma, E, R)$ [31], where the order-sorted equational theory specifies the configurations as explained above, and where R is a set of *rewrite rules* of the form $t \longrightarrow t'$ specifying the protocol concurrent transitions, that is, the protocol's "dynamics."

A set AP of atomic propositions for the configurations of a protocol $\mathcal{P} = (\Sigma, E, R)$ can be easily specified as equationally-defined Boolean predicates in (Σ, E). The atomic propositions AP, plus the choice of the sort Conf as the sort of configurations define a *Kripke structure* $\mathcal{K}(\mathcal{P})$ (see [10]), whose configurations are those of $\mathcal{P}$, whose transitions are the one-state rewrites with R modulo the equations E, and whose labeling function maps a state to all the atomic propositions that are provable true in that state. In this paper we allow deadlock configurations, and therefore the transition relation of $\mathcal{K}(\mathcal{P})$ is not required to be total.

In our modular reasoning we will use equational and rewrite theory *inclusions* $(\Sigma, E) \subseteq (\Sigma, E)$, and $(\Sigma, E, R) \subseteq (\Sigma, E, R)$, with the obvious meaning: $\Sigma \subseteq \Sigma'$, $E \subseteq E'$, and $R \subseteq R'$; and also theory *unions* $(\Sigma, E) \cup (\Sigma, E)$, and $(\Sigma, E, R) \cup (\Sigma, E, R)$, also with the obvious meaning: $\Sigma \cup \Sigma'$, $E \cup E'$, and $R \cup R'$.

3 Underlying Protocol

We use rewriting logic to express both an underlying, generic protocol satisfying minimal requirements, and the enhancement of such a protocol with DoS protection. The rewrite theories of both the underlying protocol and the enhancement of it with DoS protection shall assume that there is an equational theory $\mathcal{M} = (\Sigma_{\mathcal{M}}, E_{\mathcal{M}})$ which specifies the messages exchanged between clients and servers, with $\Sigma_{\mathcal{M}}$ a signature declaring sorts, subsorts and function symbols and $E_{\mathcal{M}}$ a set of Σ-equations. Furthermore, we make the following assumptions:

- There is a sort $\mathsf{MsgCnts}$ that describes the contents of a message. We hereafter use m, m_1, m_2, etc., as variables of sort $\mathsf{MsgCnts}$.
- There is a sort Cookie for cookies. The sort Cookie is a subsort of sort $\mathsf{MsgCnts}$. We use k, k', etc., as variables of sort Cookie.
- There is a function symbol $(\_,\_) : \mathsf{MsgCnts} \times \mathsf{MsgCnts} \rightarrow \mathsf{MsgCnts}$. Intuitively, the term (m_1, m_2) is the pair that consists of two messages m_1 and m_2.
- There is a sort Seed, a subsort of $\mathsf{MsgCnts}$, which is used by a unary function $\mathsf{rand} : \mathsf{Seed} \rightarrow \mathsf{Cookie}$, to generate a new cookie. There is a constant 1 of sort Seed and a unary function $\mathsf{next} : \mathsf{Seed} \rightarrow \mathsf{Seed}$ for incrementing the seed. As we shall see, the server wrapper stores a seed for cookie generation. The wrapper increments the seed each time it generates a fresh cookie. We shall use l, l' as variables of sort Seed. It will also be convenient to have a binary function symbol $\mathsf{Rnd} : \mathsf{Seed} \rightarrow \mathsf{MsgCnts}$ such that $\mathsf{Rnd}(l) = (\mathsf{next}(l), \mathsf{rand}(l))$.
- There are sorts CId and SId which stand for the identifiers for the clients and the servers. There is a sort Id with CId and SId as subsorts. The sort Id is a subsort of $\mathsf{MsgCnts}$. We use C, C_1, etc., as variables of sort CId, S, S_1, etc., as variables of sort SId and id, id_1, etc., to refer to variables of sort Id.
- There is a sort Msg and a ternary function symbol $(\mathsf{to}\ \_,\ \_\ \mathsf{from}\ \_) : \mathsf{Id} \times \mathsf{MsgCnts} \times \mathsf{Id} \rightarrow \mathsf{Msg}$. The term $(\mathsf{to}\ id_1, m\ \mathsf{from}\ id_2)$ stands for a message m sent by id_2 for id_1. We shall use msg, msg' as variables of sort Msg.
- There is also a constant $\mathsf{connect}$ of sort $\mathsf{MsgCnts}$. The term $(\mathsf{to}\ \mathrm{S}, \mathsf{connect}\ \mathsf{from}\ \mathrm{C})$ is the connect request sent by C to S.
- There is sort Conf along with a constant null of sort Conf. The sort Msg is a subsort of Conf. There is also a binary symbol function $\_\,;\,\_ : \mathsf{Conf} \times \mathsf{Conf} \rightarrow \mathsf{Conf}$ which intuitively stands for multiset union. Properties of associativity, commutativity and the identity (null) of $\_\,;\,\_$ are enforced in the set of equations $E_{\mathcal{M}}$. Later, we shall extend Conf to include the state of clients, servers and the intruder. A term of sort Conf will stand for protocol configurations and contain the messages, state of clients, servers and the intruder.

There might be other sorts and equations depending on the underlying protocol. For example, if the protocol uses symmetric encryption then there will be sorts for keys and

two binary function symbols, say enc and dec, for encryption and decryption respectively. Furthermore, $E_{\mathcal{M}}$ will contain the equation $\mathsf{dec}(key, \mathsf{enc}(key, m)) = m$.

The messages on the network are represented as multisets of messages. Formally, a *multiset of messages* is a ground term of sort Conf defined recursively as follows:

1. If $\mathrm{id}_1, \mathrm{id}_2$ are ground terms of sort Id and m is a ground term of sort MsgCnts then $(\mathsf{to}\ \mathrm{id}_1,\ \mathrm{m}\ \mathsf{from}\ \mathrm{id}_2)$ is a multiset of messages.
2. If B_1 and B_2 are multisets of messages then $\mathrm{B}_1; \mathrm{B}_2$ is a multiset of messages.

For example, the multiset $(\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}); (\mathsf{to}\ \mathrm{C}, \mathrm{m}'\ \mathsf{from}\ \mathrm{S})$ represents two messages – m sent by (or some entity claiming to be) the client C meant for the server S and m' sent by (or some entity claiming to be) the server S meant for the client C.

Given the message equational theory $\mathcal{M} = (\Sigma_{\mathcal{M}}, E_{\mathcal{M}})$, the underlying protocol is given using a rewrite theory $\mathcal{P} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$ with $\Sigma_{\mathcal{M}} \subseteq \Sigma_{\mathcal{P}}$ and $E_{\mathcal{M}} \subseteq E_{\mathcal{P}}$. The signature $\Sigma_{\mathcal{P}}$ in addition to $\Sigma_{\mathcal{M}}$ must contain the following sorts.

- There are sorts CIntState and SIntState which describe the internal states of the clients and servers respectively.
- There is a sort CConf, subsort of Conf, and a binary function symbol $\langle \_, \_ \rangle_c$: CId $\times$ CIntState $\rightarrow$ CConf. We use X as variable of sort CConf. For example, the term $\langle \mathrm{C}, \mathrm{CIntState} \rangle_c$ represents a client whose unique identifier is C and whose internal state is $\mathrm{CIntState}$.
- There is a sort SConf, subsort of Conf, and a binary function symbol $\langle \_, \_ \rangle_s$: SId $\times$ SIntState $\rightarrow$ SConf. We use Y as variable of sort SConf.

We shall assume that $\mathcal{P}$ ensures that the operator $(\_, \_)$: MsgCnts $\times$ MsgCnts $\rightarrow$ MsgCnts is a free constructor and that the operator $\_;\_$: Conf $\times$ Conf $\rightarrow$ Conf is a free constructor modulo associativity, commutativity and identity of null.

We distinguish ground terms that represent configurations possibly reachable in a protocol execution. The reachable configurations consist of a multiset of messages, client states and server states. Furthermore, each client and server is represented by a unique term. The set of *good configurations* is defined inductively as follows.

- null is a good configuration.
- If Conf is a good configuration and B is a multiset of messages then $\mathrm{Conf}; \mathrm{B}$ is a good configuration.
- If Conf is a good configuration, C and $\mathrm{CIntState}$ are ground terms of sort CId and CIntState respectively, then $\mathrm{Conf}; \langle \mathrm{C}, \mathrm{CIntState} \rangle_c$ is a good configuration provided Conf does not have any subterm for the form $\langle \mathrm{C}, \mathrm{CIntState}' \rangle_c$ for some $\mathrm{CIntState}'$.
- If Conf is a good configuration, S and $\mathrm{SIntState}$ are ground terms of sort SId and SIntState respectively then $\mathrm{Conf}; \langle \mathrm{S}, \mathrm{SIntState} \rangle_S$ is a good configuration provided Conf does not have any subterm of the form $\langle \mathrm{S}, \mathrm{SIntState}' \rangle_s$ for some $\mathrm{SIntState}'$.

The set of good configurations shall henceforth be called GoodConf. The execution of the protocol theory is given by a transition relation $\rightarrow_{\mathcal{P}}$. Given two good configurations Conf_1 and Conf_2, we write $\mathrm{Conf}_1 \rightarrow_{\mathcal{P}} \mathrm{Conf}_2$ iff Conf_2 can be obtained from Conf_1 by a one-step rewrite with a rule in $R_{\mathcal{P}}$.

For example, the configuration $\langle \mathrm{C}, \mathrm{CIntState}\rangle; \langle \mathrm{S}, \mathrm{SIntState}\rangle; (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C})$ represents a configuration in which a client C has sent a message m intended for server S. If the server reads the message m and changes its internal state to $\mathrm{SIntState}'$ then we shall have $\langle \mathrm{C}, \mathrm{CIntState}\rangle; (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}); \langle \mathrm{S}, \mathrm{SIntState}\rangle \rightarrow_{\mathcal{P}} \langle \mathrm{C}, \mathrm{CIntState}\rangle; \langle \mathrm{S}, \mathrm{SIntState}'\rangle$.

4 Cookie Wrapper and its Preservation Properties

Given the underlying protocol theory $\mathcal{P} = (\Sigma_{\mathcal{P}}, E_{\mathcal{P}}, R_{\mathcal{P}})$, the cookie wrapper is defined as a theory transformation $\mathcal{P} \mapsto \mathcal{W}[\mathcal{P}]$. The theory $\mathcal{W}[\mathcal{P}] = (\Sigma_{\mathcal{W}[\mathcal{P}]}, E_{\mathcal{W}[\mathcal{P}]}, R_{\mathcal{W}[\mathcal{P}]})$ extends $\mathcal{P}$ and is constructed as follows. The signature $\Sigma_{\mathcal{W}[\mathcal{P}]}$ extends $\Sigma_{\mathcal{P}}$ with the following new sorts:

1. There is a sort $\mathsf{CStoredCookiePair}$ along with a binary function symbol $(\_,\_) : \mathsf{SId} \times \mathsf{Cookie} \rightarrow \mathsf{CStoredCookiePair}$. Intuitively, the pair (S, k) will be stored by the client wrapper and it uses cookie k when sending requests to server S.
2. There is a sort $\mathsf{CStoredCookies}$ for the set of cookie pairs stored at the client site. There is a constant $\emptyset$ which stands for the empty set, along with a function symbol $\{\_\} : \mathsf{CStoredCookiePair} \rightarrow \mathsf{CStoredCookies}$ that make a term of sort $\mathsf{CStoredCookiePair}$ into a set and a function symbol $\cup$ that stands for the union operator. For example, the set $\{(\mathrm{S}, k)\} \cup \{(\mathrm{S}', k')\}$ stored at client site C means that the client C shall use the cookies k and k' while sending messages to S and S' respectively. There is also a binary function symbol $\mathsf{SIn} : \mathsf{SId} \times \mathsf{CStoredCookies} \rightarrow \mathsf{Bool}$. The function $\mathsf{SIn}(\mathrm{S}, \mathrm{CP}_c)$ returns true, if $\exists k$ s.t. $(\mathrm{S}, k) \in \mathrm{CP}_c$ and false otherwise. For example, the function $\mathsf{SIn}(\mathrm{S}_1, \{(\mathrm{S}, k)\} \cup \{(\mathrm{S}', k')\})$ returns true if and only if S_1 is either S or S'.
3. There are sorts $\mathsf{SStoredCookiePair}$ and $\mathsf{SStoredCookies}$ along with the binary function $\mathsf{CIn} : \mathsf{CId} \times \mathsf{SStoredCookies} \rightarrow \mathsf{Bool}$ for managing the cookies at server side similar to the ones at the client side.
4. There is a sort $\mathsf{WrappedCConf}$ for the wrapped client configuration along with a 4-ary function symbol $[\_\,,\_\,,\_\,,\_]_c : \mathsf{CId} \times \mathsf{CStoredCookies} \times \mathsf{Msg} \times \mathsf{CConf} \rightarrow \mathsf{WrappedCConf}$. The term $[\mathrm{C}, \mathrm{CP}_c, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), X]$ stands for the client wrapper for the client C, where CP_c is the set of cookie pairs stored at client C, m is a message (destined for server S) that is stored while a connection to S is being established, and X is the underlying protocol configuration for the client C, together with messages sent by C or addressed to C.
5. There is a sort $\mathsf{WrappedSConf}$ along with a 4-ary function symbol $[\_\,,\_\,,\_\,,\_]_s : \mathsf{SId} \times \mathsf{SStoredCookies} \times \mathsf{Seed} \times \mathsf{SConf} \rightarrow \mathsf{WrappedSConf}$ for wrapped server configurations.
6. The sorts $\mathsf{WrappedSConf}$ and $\mathsf{WrappedCConf}$ are subsorts of Conf.

The set $E_{\mathcal{W}[\mathcal{P}]}$ extends $E_{\mathcal{P}}$ with equations required in the definition of new sorts. The set $R_{\mathcal{W}[\mathcal{P}]}$ extends $R_{\mathcal{P}}$ by adding new rules for the wrapper which are given in Table 1 and discussed below.

The client wrapper rule $\mathsf{ConnectReq}$ allows the wrapper to initiate a connection request to the server S if the connection is not already established and the underlying

Client Wrapper Rules.

ConnectReq: $[C, CP_c, \mathsf{null}, (\mathsf{to}\ S, m\ \mathsf{from}\ C); X]_c \rightarrow [C, CP_c, (\mathsf{to}\ S, m\ \mathsf{from}\ C), X]_c;$
$(\mathsf{to}\ S, \mathsf{connect\ from}\ C)$ if $\mathsf{SIn}(S, CP_c) = \mathsf{false}$

SetCookie: $(\mathsf{to}\ C, k\ \mathsf{from}\ S); [C,\ CP_c, (\mathsf{to}\ S, m\ \mathsf{from}\ C)\ , X]_c \rightarrow$
$(\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C); [C,\ \{(S, k)\}\ \cup CP_c, \mathsf{null}, X]_c$
if $\mathsf{SIn}(S, CP_c) = \mathsf{false}$

MsgToServer: $[C, \{(S, k)\} \cup CP_c, \mathsf{null}, (\mathsf{to}\ S, m\ \mathsf{from}\ C); X]_c \rightarrow$
$[C, \{(S, k)\} \cup CP_c, \mathsf{null}, X]_c; (\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C)$

AcceptReply: $(\mathsf{to}\ C, m_1\ \mathsf{from}\ S); [C, CP_c, msg, X]_c \rightarrow$
$[C, CP_c, msg, (\mathsf{to}\ C, m_1\ \mathsf{from}\ S); X]_c$ if not $m_1 : \mathsf{Cookie}$

Service Wrapper Rules.

CookieGeneration: $(\mathsf{to}\ S, \mathsf{connect\ from}\ C); [S, CP_s, l, Y]_s \rightarrow (\mathsf{to}\ C, \mathsf{rand}(l)\ \mathsf{from}\ S)$
$[S, \{(C, \mathsf{rand}(\ l))\} \cup CP_s, \mathsf{next}(l), Y]_s$; if $\mathsf{CIn}(C, CP_s) = \mathsf{false}$

ResendCookie: $(\mathsf{to}\ S, \mathsf{connect\ from}\ C); [S, \{(C, k)\} \cup\ CP_s, l, Y]_s \rightarrow$
$[S, \{(C, k)\} \cup\ CP_s, l, Y]_s; (\mathsf{to}\ C, k\ \mathsf{from}\ S)$

ForwardRequest: $(\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C); [S, \{(C, k)\}\ \cup CP_s, l, Y]_s \rightarrow$
$[S, \{(C, k)\} \cup\ CP_s, l, (\mathsf{to}\ S, m\ \mathsf{from}\ C);\ Y]_s$

DropRequest1: $(\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C); [S, CP_s, l, Y]_s \rightarrow$
$[S, CP_s, l, Y]_s$ if $\mathsf{CIn}(C, CP_s) = \mathsf{false}$

DropRequest2: $(\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C); [S, \{(C, k')\} \cup\ CP_s, l, Y]_s \rightarrow$
$[S, \{(C, k')\} \cup\ CP_s, l, Y]_s$ if $(k \neq k')$

ForwardReply: $[S, CP_s, l, (\mathsf{to}\ C, m\ \mathsf{from}\ S); Y]_s \rightarrow [S, CP_s, l, Y]_s; (\mathsf{to}\ C, m\ \mathsf{from}\ S)$

Table 1. Cookie Wrapper Rewrite Rules

client wants to send a message M to S. The message M is held until the connection is established. The rule SetCookie, triggered upon receiving a cookie from S, allows the wrapper to complete the connection and release M. The cookie is stored by the wrapper and the rule MsgToServer ensures that all future service requests to S contain this cookie. The rule AcceptReply allows the wrapper to forward any future replies from S to the underlying client.

The server wrapper rule CookieGeneration, triggered when a server S receives a connection request from a client C for the first time, allows S to reply to the request with a freshly generated cookie. The cookie is stored and resent using the rule ResendCookie in reply to any further connection requests by C. The stored cookie is also used by rule ForwardRequest, which forwards service requests to S only if accompanied by the right cookie. The wrapper drops any service requests, if either the cookie mismatches or if there is no connection established, by using the rules DropRequest1 and DropRequest2 respectively. The rule ForwardRequest allows the wrapper to forward any client request (with the proper cookie) to the underlying server configuration. Finally, the rule ForwardReply allows the wrapper to forward any reply by its underlying server configuration to the client.

As in the case of a protocol theory, we need to identify the set of *good wrapped configurations* which represent the set of reachable configurations of the protocol with the DoS protection. In order to define good wrapped configurations we shall first de-

fine *well-formed unwrapped and wrapped client configurations*. A well-formed unwrapped client configuration for a client C is a ground term of sort Conf that contains a (unique) client configuration for C as well as messages sent by or sent to C. A well-formed wrapped client configuration for a client C consists of a ground term of sort $\mathsf{WrappedCConf}$ which "wraps" a well-formed unwrapped client configuration for client C. Formally,

Definition: Given a ground term C of sort CId, a *well-formed unwrapped client configuration for* C is defined recursively as follows;

1. If Cis is a ground term of sort $\mathsf{CIntState}$, then $\langle \mathrm{C}, \mathrm{Cis} \rangle_c$ is a well-formed unwrapped client configuration.
2. If Conf is well-formed unwrapped client configuration for C and m, S are ground terms of sort $\mathsf{MsgCnts}$ and SId respectively, then $\mathrm{Conf}; (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C})$ and $\mathrm{Conf}; (\mathsf{to}\ \mathrm{C}, \mathrm{m}\ \mathsf{from}\ \mathrm{S})$ are well-formed unwrapped client configurations.

If C,CP_c, S,m are ground terms of the sort $\mathsf{CId}, \mathsf{CStoredCookies}, \mathsf{SId}$ and $\mathsf{MsgCnts}$ respectively, and Conf is a well-formed unwrapped client configuration for C then $[\mathrm{C}, \mathrm{CP}_c, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), \mathrm{Conf}]_c$ is a *well-formed wrapped client configuration.*

The well-formed unwrapped and wrapped server configurations can be defined analogously to their client counterparts.

We can now identify the set of good wrapped configurations which represent possible reachable configurations of the protocol with the cookie wrapper. A good wrapped configuration consists of a multiset of messages and well-formed wrapped client and server configurations with the restriction that any given client or server appears at most once. For lack of space, we do not provide a formal definition here. The set of good wrapped configurations shall henceforth be called $\mathsf{GoodWrappedConf}$.

As before, we define the execution of the wrapped protocol by a one-step transition relation $\rightarrow_{\mathcal{W}[\mathcal{P}]} \subseteq \mathsf{GoodWrappedConf} \times \mathsf{GoodWrappedConf}$. Given two good wrapped configurations W_1 and W_2, we write $W_1 \rightarrow_{\mathcal{W}[\mathcal{P}]} W_2$ if W_2 can be obtained from W_1 by a one-step rewrite with a rule in $\mathcal{W}[\mathcal{P}]$. For example, the good wrapped configuration $[\mathrm{C}, \emptyset, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), X]_c\ ; (\mathsf{to}\ \mathrm{S}, \mathsf{connect}\ \mathsf{from}\ \mathrm{C})\ ; [\mathrm{S}, \emptyset, 1, Y]_s$ represents a configuration in which the client wrapper for C has sent a connect request to the server S. The server accepts the connect request using the rewrite rule $\mathsf{CookieGeneration}$ and hence we have $[\mathrm{C}, \emptyset, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), X]_c; (\mathsf{to}\ \mathrm{S}, \mathsf{connect}\ \mathsf{from}\ \mathrm{C}); [\mathrm{S}, \emptyset, 1, Y]_s \rightarrow_{\mathcal{W}[\mathcal{P}]} [\mathrm{C}, \emptyset, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), X]_c; (\mathsf{to}\ \mathrm{C}, \mathsf{rand}(1)\ \mathsf{from}\ \mathrm{S}); [\mathrm{S}, \{(\mathrm{C}, \mathsf{rand}(1))\}\ , \mathsf{next}(1)\ , Y]_s$.

We are almost ready to show that each execution of the wrapped protocol can be stuttering simulated by the unwrapped one. We shall however require one more definition. Consider the server wrapper rules $\mathsf{DropRequest1}$ and $\mathsf{DropRequest2}$. The server wrapper drops service requests if the cookie attached to the request mismatches the cookie stored at its site. This is not reflected in the underlying protocol as there is no such check of the service requests. However, if we consider protocols in the presence of a lossy environment, we can mimic the dropping of these requests by a message loss.

Given the equational theory $\mathcal{M} = (\Sigma_{\mathcal{M}}, E_{\mathcal{M}})$, a *lossy environment* is defined as the rewrite-theory $\mathcal{L} = (\Sigma_{\mathcal{L}}, E_{\mathcal{L}}, R_{\mathcal{L}})$, where $\Sigma_{\mathcal{L}} = \Sigma_{\mathcal{M}}$, $E_{\mathcal{L}} = E_{\mathcal{M}}$ and $R_{\mathcal{L}}$ consists of the single rewrite rule: $(\mathsf{to}\ id_1, m\ \mathsf{from}\ id_2) \rightarrow \mathsf{null}$. Similar to the way we have

defined $\rightarrow_{\mathcal{P}}$ over GoodConf and $\rightarrow_{\mathcal{W}[\mathcal{P}]}$ over GoodWrappedConf, we can define the transition relations $\rightarrow_{\mathcal{P}\cup\mathcal{L}}$ over GoodConf and $\rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}}$ over GoodWrappedConf.

4.1 Simulation

We shall show that the safety properties of the underlying protocol are preserved when we add the cookie-based DoS protection wrapper by giving a stuttering simulation map H_{fgt}: GoodWrappedConf $\rightarrow$ GoodConf between the set of good wrapped configurations GoodWrappedConf and the set of good configurations GoodConf.

The key idea in the construction of the simulation map H_{fgt} is that the connect requests and messages with cookies are used only by the DoS wrappers, and are not observed by the underlying protocols. The simulation map uses the auxiliary function h (given below), which maps a multiset of messages onto another multiset of messages by essentially forgetting all messages whose contents are only cookies or connect requests. Furthermore, in case the message consists of a cookie as the first part of the contents, the map h also forgets the cookie part. Formally,

(a) $h((\mathsf{to}\ id_1, k\ \mathsf{from}\ id_2)) = \mathsf{null}$
(b) $h((\mathsf{to}\ id_1, (k, m)\ \mathsf{from}\ id_2)) = (\mathsf{to}\ id_1, m\ \mathsf{from}\ id_2)$
(c) $h((\mathsf{to}\ id_1, \mathsf{connect}\ \mathsf{from}\ id_2)) = \mathsf{null}$
(d) $h(M) = M$ if M is not (a), (b) or (c)
(e) $h(M; X) = h(M)\,;\, h(X)$

Please note that we should read the above equations as working on equivalence classes of messages. For example, the equation $h((\mathsf{to}\ id_1, k\ \mathsf{from}\ id_2)) = \mathsf{null}$ means that any message which can be proved equal to $(\mathsf{to}\ id_1, k\ \mathsf{from}\ id_2)$ (using the set of equations E_p) also gets mapped to null. The well-definedness of equations will follow from the fact that the function $(\_\,,\,\_) : \mathsf{MsgCnts} \times \mathsf{MsgCnts} \rightarrow \mathsf{MsgCnts}$ is a free constructor and the function $\_\,;\,\_ : \mathsf{Conf} \times \mathsf{Conf} \rightarrow \mathsf{Conf}$ is a free constructor modulo associativity, commutativity and identity of null.

We are now ready to define our simulation map H_{fgt}. Please note that our description of GoodWrappedConf implies that the good wrapped configurations are multisets which contain well-formed wrapped client configurations, well-formed wrapped server configurations, and a multiset of messages. The function H_{fgt} "unwraps" the client and server configurations, and maps the multisets of messages B to $h(B)$. The messages that are held in the client wrapper waiting for the server reply are "released".

Definition: Given the set of good configurations GoodConf and good wrapped configurations GoodWrappedConf, the function H_{fgt}: GoodWrappedConf $\rightarrow$ GoodConf is defined as follows.

$$\begin{aligned} H_{fgt}\ (\ &[C^1, CP_c^1, M^1, X_c^1]_c; \ldots; [C^q, CP_c^q, M^q, X_c^q]_c; \\ &[S^1, CP_s^1, l^1, X_s^1]_s; \ldots; [S^r, CP_s^r, l^r, X_s^r]_s; B) = \\ &\qquad X_c^1, \ldots X_c^q; X_s^1, \ldots X_s^r; h(B); M^1; \ldots; \ldots M^q \end{aligned}$$

For example, the configuration $[\mathrm{C}, \emptyset, (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}), X]_c\ ;(\mathsf{to}\ \mathrm{S}, \mathsf{connect}\ \mathsf{from}\ \mathrm{C})\ ;[\mathrm{S}, \emptyset, 1, Y]_s$ in which the client wrapper for C has sent a connect request to the server S gets mapped by H_{fgt} to $X; (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}); Y$.

Please recall that the transition systems $(\mathsf{GoodConf}, \rightarrow_{\mathcal{P}})$ and $(\mathsf{GoodWrappedConf}, \rightarrow_{\mathcal{W}[\mathcal{P}]})$ describe the evolution of unwrapped and wrapped protocol configurations respectively. In order to talk about safety properties, we need to define Kripke structures. Towards this end, we shall assume that there is a set of propositions AP and a labeling function $L_{\mathcal{P}} : \mathsf{GoodConf} \rightarrow 2^{\mathsf{AP}}$ which labels the good configurations of the underlying protocol.[1]

Theorem 1 (Stuttering simulation and safety preservation) *Let $\mathcal{P}$ be a protocol rewrite theory and let $\mathcal{W}[\mathcal{P}]$ be the rewrite theory obtained by adding the cookie wrapper. Let $\mathcal{L}$ be the rewrite theory of a lossy environment. Let $\mathsf{A}_{\mathcal{P}\cup\mathcal{L}} = (\mathsf{GoodConf}, \rightarrow_{\mathcal{P}\cup\mathcal{L}}, \mathsf{L}_{\mathcal{P}\cup\mathcal{L}})$ be the Kripke structure generated from the good configurations of $\mathcal{P} \cup \mathcal{L}$ and $\mathsf{A}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}} = (\mathsf{GoodWrappedConf}, \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}}, \mathsf{L}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}})$ be the Kripke structure generated from the good wrapped configurations of $\mathcal{W}[\mathcal{P}] \cup \mathcal{L}$ such that $\mathsf{L}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}} = H_{fgt} \circ \mathsf{L}_{\mathcal{P}\cup\mathcal{L}}$.*

Then $\{(W, H_{fgt}\ (W)) | W \in \mathsf{GoodWrappedConf}\}$ is a stuttering simulation. Furthermore, given a $\mathsf{Safety}\backslash\bigcirc$ formula ψ and any $W \in \mathsf{A}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}}$,

$$H_{fgt}\ (W) \models_{\mathsf{A}_{\mathcal{P}\cup\mathcal{L}}} \psi \ \Rightarrow\ W \models_{\mathsf{A}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}}} \psi.$$

Proof. (Sketch.) We have to show that if $W_1 \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}} W_2$ then it is matched by a stuttering transition of $H_{fgt}\ (W_1)$.

The transition $W_1 \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{L}} W_2$ can be obtained from one of the three theories: (i) a one-step application of a rewrite rule in the set $R_{\mathcal{P}}$ of the underlying protocol theory, (ii) an application of the new transition rules in $R_{\mathcal{W}[P]}$, and (iii) an application of the message drop rule in $\mathcal{L}$. In the first and third cases, the transition is trivially matched. For the second case, the rules ConnectReq, MsgToServer, SetCookie, AcceptReply, ResendCookie, ForwardRequest, CookieGeneration and ForwardReply can be shown to stutter.

The more interesting cases are the rules DropRequest1 and DropRequest2. In these cases the server wrapper drops messages if the cookies mismatch (or the server doesn't have any cookie corresponding to the client). This is simulated by a message loss transition of the environment.

Formally, if W_2 is obtained from W_1 by the application of either DropRequest1 or DropRequest2, then the multiset W_1 contains a server configuration $[S, CP_s, l, SConf]_s$ and a message $t_1 = (\mathsf{to}\ S, (k, m)\ \mathsf{from}\ C)$. Furthermore, (C, k) is not in the set CP_s. The wrapped configuration W_2 is obtained from W_1 by replacing t_1 by null. By definition, $W_1 = W'; t_1$, $W_2 = W'$, $H_{fgt}\ (W_1) = H_{fgt}\ (W'); H_{fgt}\ (t_1)$ and $H_{fgt}\ (W_2) = H_{fgt}\ (W')$. Now $H_{fgt}\ (t_1)$ is a message. Due to the message loss rule in $\mathcal{L}$, we get $H_{fgt}\ (W_1) \rightarrow_{\mathcal{P}\cup\mathcal{L}} H_{fgt}\ (W_2)$. □

Please note that, due to the absence of an intruder, this result doesn't apply to cryptographic security properties of the protocol. However, it is still useful as we may be concerned with other correctness properties of the protocol expressed as safety properties. This theorem ensures that any such properties regarding the protocol are still valid when we apply the cookie wrapper.

[1] Such atomic propositions and labeling function can, for example, be equationally defined in a conservative ("protecting") extension of the protocol theory $\mathcal{P}$ (see for e.g., [10]).

5 Security with a Man-in-the-Middle Intruder

The standard assumptions of a MitM model give the intruder the ability to intercept messages, store messages, and send messages on the network. It also has the ability to apply cryptographic functions available to the protocol 'users' (for instance decrypting an encrypted message using a key it already knows). We now give the rewrite theory for a Dolev-Yao intruder, a common MitM model, and later prove a stuttering simulation between a protocol and its cookie-based wrapped version, in the presence of this intruder, ensuring the preservation of safety properties. Please note that the Dolev-Yao intruder considered herein is parametrized by an equational theory which implies that our preservation result also applies to safety properties in the presence of an intruder that can exploit algebraic properties of the constructs (such as xor and modular exponentiation) used in protocol messages.

5.1 Rewrite Theory for the Dolev-Yao Intruder

Given the equational theory $\mathcal{M} = (\Sigma_{\mathcal{M}}, E_{\mathcal{M}})$, the capabilities of the Dolev-Yao intruder are defined in terms of a rewrite theory $\mathcal{I} = (\Sigma_{\mathcal{I}}, E_{\mathcal{I}}, R_{\mathcal{I}})$. The signature $\Sigma_{\mathcal{I}}$ extends $\Sigma_{\mathcal{M}}$ and the equations $E_{\mathcal{I}}$ extend $E_{\mathcal{M}}$, with the following additional sorts and equations (recall that we already have defined sorts Conf, MsgCnts and Seed as well as symbols next, rand, Rnd in the message equational theory $\mathcal{M}$).

- There is sort SetMsgCnts along with a constant null of sort SetMsgCnts. The sort MsgCnts is a subsort of SetMsgCnts. The sort SetMsgCnts stands intuitively for a set of terms of the sort MsgCnts. There is a binary function symbol $\_;\_ : \mathsf{SetMsgCnts} \times \mathsf{SetMsgCnts} \rightarrow \mathsf{SetMsgCnts}$ which intuitively stands for set union.
- There is a sort IntruderKnowledge, subsort of Conf, along with a unary function symbol $[\_]_i : \mathsf{SetMsgCnts} \rightarrow \mathsf{IntruderKnowledge}$. Intuitively, the term $[X]_i$ stands for the intruder's knowledge, *i.e.*, the intruder knows the set of terms X. For example, the term $[key, \mathrm{m}, \mathsf{enc}(key, m)]_i$ represents an intruder who knows a plain-text message m, a key key and the encryption of m under the key key.
- There is a set of function symbols $\mathcal{F}$ which represent the cryptographic functionality available to the intruder. $\mathcal{F}$ is assumed to contain at least the 0-ary function symbol connect, and the unary function symbol $\mathsf{Rnd} : \mathsf{Seed} \rightarrow \mathsf{MsgCnts}$ which satisfies the equation $\mathsf{Rnd}(l) = (\mathsf{next}(l), \mathsf{rand}(l))$ as before.

The rewrite rules $R_{\mathcal{I}}$ given in Table 2 describe the actions that the Dolev-Yao intruder can perform. The set $E_{\mathcal{I}}$ contains the equations $E_{\mathcal{M}}$, equations that make SetMsgCnts a set, and the equation Coupling-Decoupling given in Table 2. Furthermore, $E_{\mathcal{I}}$ may contain any equations that express the algebraic properties of functions in $\mathcal{F}$.

The equation Coupling-Decoupling allows the intruder to decompose a pair into its parts and allows pairs to be formed using two terms of sort MsgCnts. The rule MsgCreation allows the intruder to send messages. The rule MsgInterception allows the intruder to intercept messages, and the rule MsgDrop allows the intruder to drop a message without storing any knowledge about its contents. Finally, the rule CryptoFunctionality allows the intruder to compute the cryptographic functions $f \in \mathcal{F}$.

Equation.
Coupling-Decoupling $[(m_1, m_2);\, Y]_i = [m_1; m_2;\, Y]_i$

Rewrite rules.

MsgCreation :	$[m; id_1; id_2; Y]_i \rightarrow [m; id_1; id_2;\, Y]_i; (\mathsf{to}\ id_1, m\ \mathsf{from}\ id)$
MsgInterception:	$[Y]_i; (\mathsf{to}\ id_1, m\ \mathsf{from}\ id) \rightarrow [m;\, Y]_i$
MsgDrop:	$[Y]_i; (\mathsf{to}\ id_1, m\ \mathsf{from}\ id) \rightarrow [Y]_i$
CryptoFunctionality:	$[m_1; \ldots; m_n;\, Y]_i \rightarrow [f(m_1, \ldots, m_n); m_1; \ldots; m_n;\, Y]_i$ if $f \in \mathcal{F}$

Table 2. Dolev-Yao Equations and Rules

Please note that the Dolev-Yao intruder is powerful enough to exploit any algebraic properties specified in the equations $E_{\mathcal{I}}$.

$\mathcal{P} \cup \mathcal{I}$ and $\mathcal{W}[\mathcal{P}] \cup \mathcal{I}$ represent the rewrite-theories for the protocol and cookie-wrapped version of the protocol extended by the Dolev-Yao intruder. Analogous to GoodConf and GoodWrappedConf defined in Section 3, we represent reachable configurations in these extended protocols by using sets GoodExtConf and GoodExtWrappedConf. Intuitively, an element of GoodExtConf stands for a possible reachable configuration of the unwrapped protocol; and is a ground term of sort Conf that represents, client and server configurations, the messages on the network, and intruder knowledge represented by a term $[I]_i$ of sort IntruderKnowledge. We assume that the identifies of all clients and servers are included in the knowledge of the intruder. Similarly, an element of GoodExtWrappedConf represents a reachable configuration of the wrapped protocol in the presence of the Dolev-Yao intruder, again represented by a term of sort IntruderKnowledge. Furthermore, we shall require that the cookie-stores in the client, server wrappers are "well-formed". That is, the cookie-stores of each client, server wrappers contain a multiset of cookie pairs of the type (Id, k), where k is a ground term of sort Cookie. We shall also require that the seed in each server is of the form, $\mathrm{l} = \mathsf{next}^n(1)$, *i.e.*, n successive applications of the function next on constant 1.

5.2 Simulation

We shall now show that the safety properties of a protocol in the presence of the Dolev-Yao intruder are preserved when we add the cookie-based DoS protection by exhibiting a stuttering simulation $H_{\mathcal{I}Sim}$: GoodExtWrappedConf $\rightarrow$ GoodExtConf between the set of good extended configurations GoodExtConf and the set of good extended wrapped configurations GoodExtWrappedConf.

Please note that the simulation map H_{fgt} used in Theorem 1, which simply drops cookies from messages, will not suffice for our purpose. This is because we need to simulate all the actions that the intruder can do when cookie-based DoS protection is being used. In particular, the intruder in the wrapped theory (say I_w represented by a ground term of sort IntruderKnowledge) can intercept messages with cookies and then generate new messages with the intercepted cookies. For example, I_w can embed the intercepted cookie within some encrypted message. The intruder in the unwrapped the-

ory (say I_u, a ground term of sort IntruderKnowledge), however does not have access to these cookies, since there is no cookie generation by the server in the unwrapped protocol. However, the intruder I_u can use its cryptographic functionality Rnd to generate cookies, which allows us to simulate all of I_w's transitions.

The simulation map, $H_{\mathcal{I}Sim}$, is defined using four auxiliary functions. The first auxiliary function, h, is the same as the one defined earlier in Section 4. The second function, ck, maps a multiset of messages to a term of sort SetMsgCnts. Given a message M, the function ck picks out a cookie if either the message contents of M is just the cookie or if the message contents of M is a pair, the first component of which is a cookie. The other functions ck_c and ck_s similarly collect cookies in CStoredCookies and SStoredCookies by picking out the cookie part of every cookie pair stored in CStoredCookies, SStoredCookies. They are formally defined as follows:

(a) $ck((\mathsf{to}\ id_1, k\ \mathsf{from}\ id_2)) = k$
(b) $ck((\mathsf{to}\ id_1, (k, m)\ \mathsf{from}\ id_2)) = k$
(c) $ck(M) = \mathsf{null}$ if M is not (a) or (b)
(d) $ck(M; X) = ck(M)\ ;\ ck(X)$
(e) $ck_c(\emptyset) = null$
(f) $ck_c(\{(C, k)\} \cup CP_c) = \{k\} \cup ck_c(CP_c)$
(g) $ck_s(\emptyset) = null$
(h) $ck_s(\{(S, k)\} \cup CP_s) = \{k\} \cup ck_s(CP_s)$

We are now ready to define our simulation map $H_{\mathcal{I}Sim}$. Please note that our description of GoodExtWrappedConf implies that the good extended wrapped configurations are multisets which contain well-formed wrapped client configurations, well-formed wrapped server configurations, an intruder state, and a multiset of messages. The function $H_{\mathcal{I}Sim}$ "unwraps" the client and server configurations, and maps the multiset of messages B to $h(B)$. The messages that are held in the client wrapper waiting for the server reply are "released". Finally, the intruder state in $H_{\mathcal{I}Sim}\ (W)$ has more facts than the intruder state in W, which allows it to mimic all actions of the intruder in W. In addition to all the facts available to the intruder in W, $H_{\mathcal{I}Sim}\ (W)$ also has cookies stored at client and server wrappers and in messages over the network. Please note that we are not giving the intruder access to the storage at these wrappers (there are no wrappers in $H_{\mathcal{I}Sim}\ (w)$). The condition is trivially satisfied at an "initial" state where the cookie stores are empty. The definition allows us to avoid an induction on reachable configurations. Furthermore, the intruder also has all the seeds that the server wrapper has, along with all the seeds that the server might have used in the past.

Definition: Given the set of good extended configurations GoodExtConf and the set of good extended wrapped configurations GoodExtWrappedConf; the function $H_{\mathcal{I}Sim}$: GoodExtWrappedConf $\rightarrow$ GoodExtConf is defined as follows.

$$
\begin{aligned}
H_{\mathcal{I}Sim}\ (\ & [C^1, CP_c^1, M^1, X_c^1]_c; \ldots; [C^q, CP_c^q, M^q, X_c^q]_c]; \\
& [S^1, CP_s^1, l^1, X_s^1]_s; \ldots; [S^r, CP_s^r, l^r, X_s^r]_s]; [Y]_i; B) = \\
& X_c^1, \ldots X_c^q; X_s^1, \ldots X_s^r; h(B); M^1; \ldots; \ldots M^q \\
& [Y; 1; \mathsf{next}(1); \mathsf{next}^2(1) \ldots; l^1; \ldots; 1; \mathsf{next}(1); \mathsf{next}^2(1) \ldots l^r; \\
& \qquad ck_c(CP_c^1); \ldots; ck_c(CP_c^q); ck_s(CP_s^1); \ldots; ck_s(CP_s^r); ck(B)]_i.
\end{aligned}
$$

For example, consider the configuration $\mathrm{Conf} = [\mathrm{C}, \{\mathrm{S}, k\}, \mathsf{null}, X_c]_c; [\mathrm{S}, \{(\mathrm{C}, k)\}, \mathsf{next}(1), X_s]_s; (\mathsf{to}\ \mathrm{S}, (k, \mathrm{m})\ \mathsf{from}\ \mathrm{C})\ ; [\mathrm{C}, \mathrm{S}]_i$ in which the client has established a connection with the server S (with cookie k) and has sent a service request. The intruder has the identities of the client and the server. By definition, $H_{\mathcal{I}Sim}$ "unwraps" the client, the server, the service request, and stores the cookie k in the intruder memory. Formally, $H_{\mathcal{I}Sim}\ (\mathrm{Conf}) = X_c; X_s; (\mathsf{to}\ \mathrm{S}, \mathrm{m}\ \mathsf{from}\ \mathrm{C}); [\mathrm{C}, \mathrm{S}, 1, \mathsf{next}(1), k]_i$.

We have the second result of the paper.

Theorem 2 *Let $\mathcal{P} \cup \mathcal{I}$ be the rewrite theory for the protocol $\mathcal{P}$ in the presence of a Dolev-Yao intruder, and $\mathsf{A}_{\mathcal{P}\cup\mathcal{I}} = (\mathsf{GoodExtConf}, \rightarrow_{\mathcal{P}\cup\mathcal{I}}, \mathsf{L}_{\mathcal{P}\cup\mathcal{I}})$ be the Kripke structure generated from the good extended configurations. Let $\mathcal{W}[\mathcal{P}] \cup \mathcal{I}$ be the rewrite theory for the cookie wrapped protocol $\mathcal{W}[\mathcal{P}]$ in the presence of a Dolev-Yao intruder, and $\mathsf{A}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}} = (\mathsf{GoodExtWrappedConf}, \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}}, \mathsf{L}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}})$ be the Kripke struture generated from the good extended wrapped configurations, such that, $L_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}} = H_{\mathcal{I}Sim} \circ \mathsf{L}_{\mathcal{P}\cup\mathcal{I}}$.*

Then for any $\mathsf{Safety}\backslash\bigcirc$ formula ψ and any $W \in \mathsf{A}_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}}$,

$$H_{\mathcal{I}Sim}\ (W) \models_{\mathsf{A}_{\mathcal{P}\cup\mathcal{I}}} \psi \ \Rightarrow\ W \models_{\mathsf{A}_{\mathcal{W}[P]\cup\mathcal{I}}} \psi.$$

Proof. (Sketch.) It suffices to show that $\{(W, H_{\mathcal{I}Sim}\ (W)) | W \in \mathsf{GoodWrappedConf}\}$ is a stuttering simulation, *i.e.*, whenever, $W_1 \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}} W_2$, then it is matched by a stuttering transition of $H_{\mathcal{I}Sim}\ (W_1)$.

The transition $W_1 \rightarrow_{\mathcal{W}[\mathcal{P}]\cup\mathcal{I}} W_2$ can either be a transition obtained by a one-step application of a rewrite rule of the underlying protocol theory $\mathcal{P}$, or a transition obtained by an application of the new transition rules of the wrapper $\mathcal{W}$, or a transition obtained by an application of the transition rules in the intruder theory. In the first case, the transition is trivially matched. For the second case, the rules ConnectReq, MsgToServer, SetCookie, AcceptReply, ResendCookie, ForwardRequest and ForwardReply can be shown to stutter. The rules DropRequest1, DropRequest2 can be simulated by the MsgDrop rule of the intruder.

If W_2 is obtained from W_1 by the application of CookieGeneration resulting in some server wrapper generating a cookie k, then this step is matched by the intruder using the its crypto functionality Rnd to generate the same cookie k.

Now, assume that W_2 is obtained from W_1 by the application of a rule in $\mathcal{R}_{\mathcal{I}}$. Please observe that the set of facts that the intruder possesses in W_1 is contained in the set of facts that the intruder possesses in $H_{\mathcal{I}Sim}\ (W_1)$. Hence, "crypto functionality" can be simulated exactly. Message creation of M can be simulated by "message creation" of $h(M)$. Please note that $h(M)$ creation requires less information. The "message interception" of M is simulated by interception of $h(M)$. The missing information (cookie) is already in the possession of the intruder in $H_{\mathcal{I}Sim}\ (W_1)$. □

5.3 Modular reasoning with cookies in IKEv2

In this section we describe how the cookie-wrapper can be applied to a practical protocol vulnerable to a DoS attack. Using the results in this paper, we can show that any $\mathsf{Safety}\backslash\bigcirc$ properties that have been verified on the original protocol (in the presence

of a Dolev-Yao intruder) still apply to the cookie-wrapper version of the protocol (in the presence of a Dolev-Yao intruder). In effect, we get the added benefit of some DoS protection without having to prove earlier safety properties again.

IPSec is a suite of protocols used to provide authentication and encryption on top of IP. The Internet Key Exchange (IKE) protocol is a part of the IPSec suite that allows two communicating parties to, among other things, securely generate session keys as well as exchange cryptographic parameters. IKEv2 ([24]) is the current version of the protocol. Since IPSec runs on top of the *connectionless* protocol UDP, there is no guarantee that an initiation request for an IKE session from a client has come from a valid IP address. This creates a DoS vulnerability as an attacker can send a large number of IKE initiation requests to a server with spoofed IP addresses. In fact, an earlier version of IKE (IKEv1[34, 29, 22]) was susceptible to this attack and protecting against such an attack was part of the motivation for IKEv2.

AVISPA ([6]) is an automated tool for verification of Internet protocols and applications. It has been used to verify various security protocols in the IPSec suite including the variants of IKEv2. In particular, it has been used to prove the secrecy of session keys exchanged in the key-exchange sub-protocol (IKEv2-DS[6]) which is vulnerable to a DoS attack. It follows from our results in this paper that the cookie-enhanced version of this protocol has the same guarantees.[2]

6 Related Work

Modular specification and reasoning has attracted a lot of attention in formal analysis of distributed systems and communication protocols. The advantage of this approach is to allow verification of individual sub-systems without worrying about *all* interactions between different layers. Towards this end, the "onion skin" modular model of actor reflection has been proposed in [2, 13, 32]. Our formalization of the cookie-based DoS protection mechanism can be seen as an instance of this formalism in which we are trying to develop the program we discussed in [3]. Ultimately we would like to be able to provide a formal modular treatment of protocols like Adaptive Selective Verification (ASV) based on the shared channel model [20, 25].

The modular framework for reasoning about a stack of protocols for security properties is less common. The security protocols are generally specified and analyzed individually. Recently, a modular proof of correctness (under Dolev-Yao assumptions) of the IEEE 802.11i and TLS suite of protocols was given in [23] using the Protocol Composition Logic [14]. Formal analysis of the VoIP protocol stack is carried out in [21] and the tunnel calculus is used to model security tunnels in stacks in [19]. Finally, there is a large body of work on compositional reasoning of cryptographic protocols. The primary focus of this line of research is to show that if some protocol is secure under the Dolev-Yao assumption of black-box cryptography, then the protocol remains secure when the black-box cryptographic functions are instantiated with their cryptographic

[2] Note that IKEv2 has its own cookie-based mechanism for prevention of such an attack which has a slightly different implementation. Therefore, our claim applies to a modular wrapper-based implementation of cookies.

realizations (see, *e.g.* [7]). Compositional reasoning is also the technique used in Protocol Composition Logic [14, 11], where, the correctness of a single protocol is derived by proving assertions about individual actions of honest participants. These compositional reasoning principles focus on safety properties and therefore complement the modular reasoning methods explored in this paper, which focus on the preservation of safety properties when DoS protection mechanisms are added.

The availability properties of the cookie-based mechanism is not the focus of our work, but there are several works in the literature which analyze availability properties. A general cost-based framework for analyzing protocols for DoS resistance has been proposed in [30] and is based on the fail-stop model in [18]. DoS-resistance is stated using an information flow property in a cost-based framework in [26]. Observation equivalence and a cost-based framework is used in [1] to analyze the availability properties of the JFK protocol. Other works on formal analysis of availability properties use branching-time logics [37, 28] and continuous stochastic logic [4].

Finally, we observe that rewriting logic has a well-established tradition in specification and analysis of security protocols [9, 33, 15, 12].

7 Conclusions and Future Work

We have proposed a modular approach to both the specification of DoS protection mechanisms as generic wrappers, and the preservation of safety properties of protocols when hardened by such wrappers. We have discussed how a cookie-based mechanism can be specified this way; and how safety properties of the underlying protocol are preserved by the cookie wrapper with and without the presence of a Dolev-Yao intruder.

This work represents a step toward capabilities we described in [4]. In that work we showed how a DoS-hardened protocol could be formally analyzed for its DoS protection capabilities by examining probabilistic guarantees for selective verification [20], a DoS protection mechanism based on the use of bandwidth limitations. The current work shows how to prove invariant properties between the MitM model and a model for DoS protection. However, the target DoS protection model is non-probabilistic (based on cookies). A next step along this path is to show this invariance for a system in which the DoS protection mechanism is probabilistic as in [4]. As an intermediate step, the results presented here about preservation of safety properties should be extended to preservation of properties definable in the more general logic $ACTL^*$ minus the next operator. Another intermediate step is to more completely formalize a protocol like IKEv2, so that the results of the paper can be more completely applied. A more ambitious topic worth investigating is the verification of *generic availability properties* of a given wrapper. That is, given a wrapper $\mathcal{W}$, can we reason about the availability properties that $\mathcal{W}$ will provide for *any* protocol $\mathcal{P}$ under minimal assumption on $\mathcal{P}$? Yet another topic for future research is the development of automated formal reasoning methods for proving preservation of a given class of properties of an underlying protocol $\mathcal{P}$ when such a protocol is composed with a given wrapper $\mathcal{W}$.

Acknowledgments The authors thank Catherine Meadows, Omid Fatemieh, and Fariba Khan for their suggestions. We also benefited from comments by anonymous review-

ers. Rohit Chadha was supported in part by NSF CCF04-29639 and NSF CCF04-48178. Carl Gunter was supported in part by NSF CNS05-24516 CNS05-5170 CNS05-09268 CNS05-24695 CNS07-16421, DHS 2006-CS-001-000001, ONR N00014-04-1-0562 N00014-02-1-0715, and a grant from the MacArthur Foundation. Jose Meseguer was supported in part by NSF CNS05-24516 and NSF CNS07-16638. Ravinder Shankesi was supported in part by NSF CNS05-24516. Mahesh Viswanathan was supported in part by NSF CCF04-48178 and NSF CCF05-09321. Views expressed in the paper are those of the authors.

References

1. M. Abadi, B. Blanchet, and C. Fournet. Just Fast Keying in the Pi Calculus. In *Programming Languages and Systems: 13th European Symposium on Programming*, volume 2986 of *LNCS*, pages 340–354, 2004.
2. G. Agha, S. Frolund, R. Panwar, and D. Sturman. A Linguistic Framework for Dynamic Composition of Dependability Protocols. *IEEE Parallel and Distributed Technology: Systems and Applications:*, 1:3–14, 1993.
3. G. Agha, M. Greenwald, C. A. Gunter, S. Khanna, J. Meseguer, K. Sen, and P. Thati. Formal modeling and analysis of dos using probabilistic rewrite theories. In *Foundations of Computer Security (FCS '05)*, 2005.
4. G. Agha, C. Gunter, M. Greenwald, S. Khanna, J. Meseguer, K. Sen, and P. Thati. Formal Modeling and Analysis of DoS using Probabilistic Rewrite Theories. In *Workshop on Foundations of Computer Security*, 2005.
5. B. Alpern and F.B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, 1985.
6. A. Armando et al. The Avispa Tool for the Automated Validation of Internet Security Protocols and Applications. In *17th Int. Conf. on Computer Aided Verification*, volume 3576 of *LNCS*, pages 281–285, 2005.
7. M. Backes, B. Pfitzmann, and M. Waidner. A Composable Cryptographic Library with Nested Operations. In *10th ACM conference on Computer and Communications Security*, pages 220–230, 2003.
8. D. J. Bernstein. SYN cookies. http://cr.yp.to/syncookies.html, 1996.
9. I. Cervesato, N. A. Durgin, P. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12-th IEEE Computer Security Foundations Workshop*, pages 55–69, 1999.
10. M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude – A High-Performance Logical Framework*, volume 4350. LNCS, 2007.
11. A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. A Derivation System and Compositional Logic for Security Protocols. *J. Comput. Secur.*, 13(3):423–482, 2005.
12. G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Workshop on Formal Methods and Security Protocols*, 1998.
13. G. Denker, J. Meseguer, and C. Talcott. Rewriting Semantics of Meta-Objects and Composable Distributed Services. In *3rd. Intl. Workshop on Rewriting Logic and its Applications*, 2000.
14. N. Durgin, J. C. Mitchell, and D. Pavlovic. A Compositional Logic for Proving Security Properties of Protocols. *J. Comput. Secur.*, 11(4):677–721, 2004.
15. S. Escobar, C. Meadows, and J. Meseguer. A Rewriting-Based Inference System for the NRL Protocol Analyzer: Grammar Generation. In *Formal Methods in Security Engineering*, pages 1–12, 2005.

16. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *RFC 2068: Hypertext Transfer Protocol – HTTP/1.1*. Internet Society, 1997.
17. J.A. Goguen and J. Meseguer. Order-sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
18. L. Gong and P. Syverson. Fail-stop Protocols: An Approach to Designing Secure Protocols. In *5th International Working Conference on Dependable Computing for Critical Applications*, pages 44–55, 1995.
19. A.E. Goodloe and C. A. Gunter. Reasoning about Concurrency for Security Tunnels. In *IEEE Computer Security Foundations (CSF '07)*, 2007.
20. C.A Gunter, S. Khanna, K. Tan, and S.S. Venkatesh. DoS Protection for Reliably Authenticated Broadcast. In *Network and Distributed System Security Symposium*, 2004.
21. A. Gupta and V. Shmatikov. Security Analysis of Voice-over-IP Protocols. *Computer Security Foundations Symposium*, 00:49–63, 2007.
22. D. Harkins and D. Carrel. *The Internet Key Exchange(IKE)*. Internet Society, 1998.
23. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A Modular Correctness Proof of IEEE 802.11i and TLS. In *12th ACM conference on Computer and Communications Security*, pages 2–15, 2005.
24. C. Kaufman. *RFC 4306: Internet Key Exchange (IKEv2) Protocol*. Internet Society, 2005.
25. S. Khanna, S. S. Venkatesh, O. Fatemieh, F. Khan, and C. A. Gunter. Adaptive Selective Verification. In *IEEE Conference on Computer Communications (INFOCOM '08)*, 2008.
26. S. Lafrance and J. Mullins. An Information Flow Method to Detect Denial of Service Vulnerabilities. *Journal of Unified Computer Science*, 9(11):1350–1369, 2003.
27. O. Lichtenstein, A. Pnueli, and L.D. Zuck. The Glory of the Past. In *Conference on Logic of Programs*, pages 196–218, 1985.
28. A. Mahimkar and V. Shmatikov. Game-based Analysis of Denial-of-Service Prevention Protocols. In *18th IEEE workshop on Computer Security Foundations*, pages 287–301, 2005.
29. D. Maughan, Schertler, M. M., Schneider, and J. Turner. *Internet Security Association and Key Management Protocol(ISAKMP)*. Internet Society, 1998.
30. C. Meadows. A Formal Framework and Evaluation Method for Network Denial of Service. In *12th IEEE workshop on Computer Security Foundations*, 1999.
31. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
32. J. Meseguer and C. Talcott. Semantic Models for Distributed Object Reflection. In *16th European Conference on Object-Oriented Programming*, pages 1–36. Springer-Verlag, 2002.
33. J. Mitchell N. Durgin, P. Lincoln and A. Scedrov. Multiset Rewriting and the Complexity of Bounded Security Protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
34. D. Piper. *The Internet IP Security Domain Of Interpretation for ISAKMP*. Internet Society, 1998.
35. 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 *IEEE Symposium on Security and Privacy*, pages 208–223, 1997.
36. A.P. Sistla. Safety, Liveness and Fairness in Temporal Logic. *Formal Asp. Comput.*, 6(5):495–512, 1994.
37. C.-F. Yu and V.D. Gligor. A Specification and Verification Method for Preventing Denial of Service. *IEEE Transactions on Software Engineering*, 16(6):581–592, 1990.