Using Coloured Petri Nets to Simulate DoS-resistant protocols

Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto

Information Security Institute
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia
s.tritilanunt@student.qut.edu.au,
{c.boyd, e.foo, j.gonzaleznieto}@qut.edu.au

Abstract. In this work, we examine unbalanced computation between an initiator and a responder that lead to resource exhaustion (one type of denial-of-service, DoS) attacks in key exchange protocols. We construct two models; one is the well-known Internet protocol named Secure Socket Layer (SSL) protocol, and the other one is the Host Identity Protocol (HIP) which has built-in DoS-resistant mechanisms. To examine such protocols, we develop a formal framework based on Timed Coloured Petri Nets (Timed CPNs) and use a simulation approach provided in CPN Tools to achieve a formal analysis. By adopting the key idea of Meadows' cost-based framework and refining the definition of operational costs during the protocol execution, our simulation provides an accurate cost estimate of protocol execution comparing among those principals, as well as the percentage of successful connections from legitimate users under four different strategies of DoS attack.

1 Introduction

Denial-of-service (DoS) attacks continue to be one of the most troublesome security threats to communication networks. DoS attacks can be classified roughly into two types: *flooding* attacks and *logical* attacks. During a flooding attack the adversary simply keeps sending messages to the victim so that the victim is unable to process any genuine requests for service. Logical attacks try to be more clever and aim to exhaust either the computational or memory resources of the victim by exploiting some feature of the communications protocol.

A general method to defend against logical DoS attacks is to authenticate connections before committing significant resources to servicing the connection. In practice, though, secure authentication is a computationally expensive process and so the effort expended in authenticating has the potential to be turned into a DoS attack in itself. Recognising this dilemma, protocol designers in the 1990s advocated a simplified form of authentication to be used before full-fledged cryptographic authentication takes place. A canonical example of this is the use of *cookies* first suggested by Karn and Simpson [18]. This mechanism can be recognised as the principle of *gradual authentication* [25] in which the server uses multiple authentication mechanisms, each successive one being more computationally expensive. Such methods have been incorporated into several protocols for authentication and key exchange, most notably into the widely deployed IPSec protocols [1].

Design of key exchange protocols has long been considered a delicate problem, but the analysis becomes even harder when DoS prevention is an additional requirement. Meadows [19] introduced a systematic framework to analyse DoS resistance by computing and comparing the cost incurred by both parties at each step in a (key exchange) protocol. Meadows analysed the STS protocol (a protocol without special DoS resistance properties) and later Smith et al. [24] used Meadows' framework to analyse the JFK protocol [1] in order to demonstrate its DoS prevention capabilities.

Surprisingly, there has been little interest in the research community in applying Meadows' framework to different protocols. Moreover, the limited application so far has suffered from two significant shortcomings which make the results of restricted value.

1. The cost analysis has only taken into account *honest* runs of the protocol. In principle, the adversary (typically the client in a client-server protocol) can deviate arbitrarily from the protocol in order to achieve an attack. By only taking into account honest behaviour it is quite likely the logical attacks will be missed. While Meadows certainly recognised this fact, no research has yet examined the effectiveness of the framework in detecting such potential attacks.

2. Meadows used only a coarse measure of computational cost, with three levels denoted as cheap, medium or expensive. In practice it can be quite difficult to classify and compare operations in such a limited way. For example, in Meadows' classification digital signature generation and verification are counted as of equal cost, yet in practice an RSA signature generation may take 2 or 3 order of magnitude more effort than RSA signature verification.

Motivated by the above two limitations, this paper provides a refinement of Meadows' cost-based framework. For our sample protocols we use the Host Identity Protocol (HIP) [21], which has built-in DoS resistance, and compare it with the well-known Secure Socket Layer (SSL) protocol. To develop a formal framework of such protocols, we use CPN Tools [27] which is a general-purpose verification tool for modeling and analysing Coloured Petri Nets. Using CPNs as our formalism, we provide a formal specification of two protocols to allow automatic searching of adversary and victim cost under different adversarial attack strategies. Moreover, we set up another experiment for examining the tolerance of HIP under such attacks.

Comparing to the previous work on the analysis of HIP by Beal and Shepard [6] that employs a mathematical approach, simulation approaches are also valued in the research community since they have been applied not only for exploring vulnerabilities in cryptographic protocols, but also guaranteeing security services of such protocols. Using simulation approaches has several benefits over mathematical analysis; for instance, simulation provides *flexibility* to the developer to adjust parameters for evaluating the system. Simulation also provides *visualization* to users who can see and learn what is happening during the simulation of cryptographic protocols to gain more understanding for evaluating the correctness of those protocols.

The main contributions of this paper are:

- a refinement of Meadows' cost-based framework to more accurately represent the cost of typical cryptographic algorithms;
- the first formal specification and automatic analysis of Meadows' framework;
- a cost-based model of SSL;
- a cost-based model of HIP protocol in Timed Coloured Petri Nets (Timed CP-Nets);
- simulation and analysis of HIP under normal conditions and under four scenarios of DoS attacks.

2 Background and Previous Work

The purposes of Section 2 are to provide the background on the Meadows's cost-based framework, SSL and HIP protocol, as well as the previous work on the analysis of security protocols using Coloured Petri Nets.

2.1 Meadows's Cost-Based Framework

Meadows framework [19] works by comparing cost to the attacker and cost to the defender, defined using a *cost set*. To model the protocol framework, we need to calculate the cost of the sequence of protocol actions, comparing between the attacker and the defender. Once the actions of each protocol principal are classified into the computational costs cheap, medium, or expensive, all actions of the protocol run can be compared. The protocol is secure against DoS attacks, if the final cost is great enough from the point of view of the attacker in comparison with the cost of engaging in the events up to an accepted action from the point of view of the defender. Otherwise, we conclude that the protocol is insecure against DoS attacks.

Considering the characteristic of DoS attacks, there are two possible ways mentioned by Meadows [19] to cause the defender to waste resources. First, the defender may process a bogus instance of a message inserted by the attacker into a protocol. The cost to an attacker is the cost of creating and inserting the bogus message, while the cost to the defender is the cost of processing the bogus message until an attack is detected. Second, the defender participates in a protocol execution with bogus instances of the attacker. The cost of this situation is equivalent to the cost of running the entire protocol until the defender can detect the attack or the attack stops.

At this stage, we limit the abilities of an attacker during the protocol execution to take one of a small number of possible actions when the protocol specifies that a message should be sent: the attacker either continues normally with the protocol or partially completes the protocol. Intuitively this is the most obvious ways for an adversary to make the defender use unwanted resources. In our examples, the adversary sends messages at two points in the protocol; either attack the first message by flooding a large number of random messages to overwhelm the resources of the responder, or attack its second message by faking its packets to waste the responder resources for verifying it.

2.2 Protocol Notation

For the protocols presented in this section, we focus only important elements for the simplification of the protocol description. Any data such as header information that are not relevant to the discussion of DoS resistance are omitted. For complete descriptions of the protocols, the reader is referred to the full protocol specifications. The protocol notation used for the remainder of this section are presented in Table 1.

Table 1. Protocol Notation

Messages	**Notation**
I	The principal who initiates the request message known as Initiator or Client
R	The principal who responds to the request message known as Responder or Server
$H(M)$	Unkeyed cryptographic hash of the message M
$H_{K_{(\cdot)}}(M)$	Keyed cryptographic hash of the message M, with key $K_{(\cdot)}$
$E_{K_{(\cdot)}}\{M\}$	Symmetric encryption of message M with the secret key $K_{(\cdot)}$
$D_{K_{(\cdot)}}\{M\}$	Symmetric decryption of message M with the secret key $K_{(\cdot)}$
$PK_R[M]$	Asymmetric encryption of the message M by the public key PK_R belonging to R
$SK_R[M]$	Asymmetric decryption of the message M by the private key SK_R belonging to R
$Sig_I(\cdot)$	Digital signature signed by the private key SK_I belonging to the principal I
$Sig_R(\cdot)$	Digital signature signed by the private key SK_R belonging to the principal R
$LSB(t, k)$	Returns the k least significant bits of an output by taking a string t as input
0^k	A string consisting of k zero bits
p, q	Large prime numbers
i, r	A Diffie-Hellman secret parameter of I and R, respectively
g	Group generator of order q used in Diffie-Hellman key exchange and key agreement protocol
s	A periodically changing secret only known to the responder R
K_s	A session key generated by key establishment protocol used to secure ongoing communications
HIT_I, HIT_R	The host identity tag of I and R created by taking a hash over the host identity HI_I and HI_R
$Cert_R$	A certificate which contains a responder's identity and a public key used for authentication

2.3 Secure Socket Layer (SSL)

Secure Sockets Layer (SSL) is a well-known Internet protocol developed by Netscape [12] for establishing and transmitting secure data over the Internet. In order to establish a secure communication, an initiator and a responder have to negotiate the cryptographic algorithms and optionally authenticate each other by using public-key cryptosystems. SSL uses public-key encryption techniques not only for the mutual authentication, but also for the protection of a session key generated during the SSL protocol handshake. This session key is a short-term symmetrical key generated by an initiator and temporarily used during the secure communication. The description of the SSL handshake is illustrated in Figure 1.

I		R
1) create *ClientHello*	$\xrightarrow{\textit{Client Hello}}$	create $Cert_R$
	$Cert_R, g^r,$	choose $r \in_R [1, 2, \ldots, q-2]$
2) verify $Cert_R$	$\xleftarrow{\textit{ServerHelloDone}}$	compute g^i
choose $i \in_R [1, 2, \ldots, q-2]$		
compute g^i		
compute $K_s = DH(g^{ir})$		
3) encrypt $PK_R[K_s]$	$\xrightarrow{PK_R[K_s]}$	decrypt $SK_R[K_s]$
4) verify *HMAC*	$\xleftarrow{HMAC}$	compute $HMAC = H_{K_s}(Data)$

Fig. 1. SSL Protocol: DH Key Agreement without Client Authentication Mode is used

To provide an explanation of the SSL handshake, we shall use the Diffie-Hellman key agreement without client authentication mode as an example because it is the most simplest mode (most general users do not have a certificate). At the beginning of the SSL handshake, the initiator sends a *Client Hello* message to the responder for establishing a secure connection. Following, the responder responds with its certificate ($Cert_R$), and Diffie-Hellman key agreement parameter. Now the responder will send the *ServerHelloDone* message indicating that the responder completes the hello-message phase. The server will then wait for the initiator response.

In order to reply the responder, the initiator has to send the *client key exchange* message which contains a session key (K_s). This session key is generated by using Diffie-Hellman key agreement protocol, $DH(g^{ir})$. In order to send this key to the responder in a secure manner, the initiator uses the responder's public key (PK_R) provided in the responder's certificate to encrypt it; $PK_R[K_s]$. At this point, a third message is completely done by the initiator.

To verify the initiator's message, the responder starts to decrypt the message; $SK_R[K_s]$, by using a private key SK_R to obtain a session key K_s. If this message is valid, the responder generates hashed-MAC ($HMAC$) by using K_s, and sends it to the initiator for performing a key confirmation. At this point, the SSL handshake is complete and the initiator and responder begin to exchange secure information.

2.4 Host Identity Protocol (HIP)

The host identity protocol (HIP) has been developed by Moskowitz [21]. Later, Aura et al. [3] found some vulnerabilities and proposed guidelines to strengthen the security of HIP. The base exchange of HIP is illustrated in Figure 2.

HIP adopts a proof-of-work scheme proposed by Jakobsson and Juels [15] for countering resource exhaustion attacks. In a proof-of-work, HIP extends the concept of a *client puzzle*, first proposed by Juels and Brainard [17], and later implemented by Aura et al. [5] for protecting the responder against DoS attacks in authentication protocols. Moreover, HIP allows the additional feature of the client puzzle that helps the responder to delay state creation [4] until the checking of the second incoming message and the authentication has been done in order to prevent the responder against resource exhaustion attack.

2.5 Coloured Petri Nets

Coloured Petri Nets (CPNs) [9,16] are one type of high-level nets based on the concept of Petri Nets developed back in 1962 by Petri [23]. CPNs is a state and action oriented model which consists of places, transitions, and arcs.

I		R
		Precomputed parameters choose $r, s \in_R [1, 2, \ldots, q-2]$ sign $sig_{R1} = Sig_R(g^r, HIT_R)$ ....
1) create HIT_I, HIT_R	$\xrightarrow{HIT_I, HIT_R}$	check HIT_R compute $C = LSB(H(s, HIT_I, HIT_R), 64)$
2) verify sig_{R1} Find J such that $LSB(H(C, HIT_I, HIT_R, J), k) = 0^k$ choose $i \in_R [1, 2, \ldots, q-2]$ compute $K_e = H(HIT_I, HIT_R, g^{ir}, 01)$ encrypt $E1 = E_{K_e}\{HI_I\}$ sign $sig_I = Sig_I(HIT_I, HIT_R, J, g^i, E1)$	$\xleftarrow[puzzle, g^r, sig_{R1}]{HIT_I, HIT_R,}$	choose $k \in [0, 1, \ldots, 40] \rightarrow puzzle = (C, k)$
3)	$\xrightarrow[J, g^i, E1, sig_I]{HIT_I, HIT_R,}$	compute $C = LSB(H(s, HIT_I, HIT_R), 64)$ check $LSB(H(C, HIT_I, HIT_R, J), k) \overset{?}{=} 0^k$ compute $K_e = H(HIT_I, HIT_R, g^{ir}, 01)$ decrypt $E1$ verify sig_I compute $K_s = H(HIT_I, HIT_R, g^{ir}, 02)$ compute $HMAC = H_{K_s}(HIT_I, HIT_R)$
4) verify sig_{R2} compute $K_s = H(HIT_I, HIT_R, g^{ir}, 02)$ check $H_{K_s}(HIT_I, HIT_R) \overset{?}{=} HMAC$	$\xleftarrow[HMAC, sig_{R2}]{HIT_I, HIT_R,}$	sign $sig_{R2} = Sig_R(HIT_I, HIT_R, HMAC)$

Fig. 2. HIP Protocol [21]

Over many years, cryptographic and security protocols have been modeled and verified using Coloured Petri Nets [11, 14, 20, 22]. Neih and Tavares [22] implemented models of cryptographic protocols in the form of Petri Nets. In order to explore vulnerabilities of such protocols, they allowed an implicit adversary with limited abilities to launch attacks and then examined the protocol using exhaustive forward execution technique. Doyle [11] developed a model of three-pass mutual authentication and adopted the forward state-space searching technique from Neih and Tavares. In addition to Neih and Tavares's model, Doyle allowed more sophisticated abilities of an adversary; multiple iteration and parallel session attacks. Han [14] has adopted CPNs specification for constructing a reachability graph to insecure states and examining the final states in OAKLEY and the Open Network Computing Remote Procedure Call (ONC RPC) protocol. In 2004, Al-Azzoni [2] has developed a model of Needham-Schroeder public key authentication protocol and Tatebayashi-Matsuzaki-Neuman (TMN) key exchange protocol. In this work, Al-Azzoni has introduced the new scheme to reduce the size of the occurrence graph for obtaining the practical result in the CPN programming called Design/CPN tool. By examining a reachability test to insecure states, Al-Azzoni has found several flaws of such protocols, for instance, an adversary is able to impersonate a legitimate user during the protocol run without knowledge from such legitimate user.

To the best of our knowledge, there is no implementation of CPNs focusing on an exploration of vulnerabilities based on unbalanced computation that might lead to DoS attacks in key exchange protocols.

3 Modeling Cryptographic Protocols with Coloured Petri Nets

We briefly explain the description of a Coloured Petri Nets representation for constructing CPN models. The abilities of an individual adversary used in the simulation are also provided in this section.

3.1 CPNs Objects Description

Prior to demonstrate the model of cryptographic protocols, we shall describe the concept of fundamental elements provided in the CPN Tools for constructing our cryptographic protocols. To implement a model relating with the concept of states and actions as illustrated in Figure 3, some important constructions of Timed CPNs including:

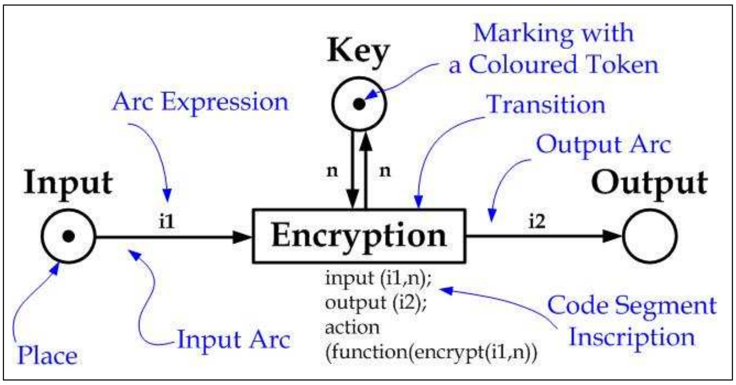


Fig. 3. An Example of CPNs Constructions

1. *Place*: is drawn as ellipse or circle which is represented states of a system. Each place has a colour set which determines the type of data that the place can carry, i.e. type of users, type of messages.
2. *Type*: is used to specify the colour set of a token in each place. For example, a *User* colour set in our protocol consists of an honest client (`hc`), four types of adversary (`ad1-4`), and a responder/server (`sv`).
3. *Marking*: is a state of a Timed CPNs. It consists of a number of tokens positioned on the individual places. The marking of a place can be a multi-set of token values, for example, incoming requested packets from multiple honest clients and different types of adversaries arrive to the same server.
4. *Transition*: represents actions of a system, which is drawn as rectangles. A transition is connected with input places by incoming arcs and output places by outgoing arcs. Some examples of transition in our model are hash function and encryption algorithms.
5. *Arc*: is used to connect transitions and places. Each arc contains a weight value called an arc expression which represents the number of removed token from input places traveling to output places.
6. *Arc Expression*: determines the number of tokens which are removed from the input place and added to the output place during occurrences of transitions. This action represents a dynamic behaviour which can be seen as the traversing of messages in the cryptographic protocols.

3.2 Adversary's Ability

In the simulation, our goal is to explore unbalanced computational vulnerabilities in the DoS-resistant protocols. As a result, not only the honest client (`hc`) who initiate the legitimate traffic, and the responder to participate in the protocol execution, but also we allow four types of adversary who have the similar goal to deny the service of the responder by overwhelming the responder's resource. The major different of individual adversary is:

Type 1 adversary (`ad1`) computes a valid first message (may be pre-computed in practice), and then takes no further action in the protocol. This type of adversary is used in the protocol simulation for both SSL and HIP.

Type 2 adversary (`ad2`) completes the protocol normally until the third message is sent and takes no further action after this. Type 2 adversary is used only in HIP in order for investigating the effect of the client puzzle. The computations of this adversary include searching a correct client puzzle solution J, generating a session key K_e and encrypting a public key PK_I, and finally computing a digital signature Sig_I.

Type 3 adversary (`ad3`) completes the protocol step one and two with the exception that the adversary does not verify the responder signature sig_{R1}. The adversary searches for a correct client puzzle solution J but randomly chooses the remaining message elements: an encrypted element $K_e\{HI_I\}$ and a digital signature sig_I. The adversary takes no further action in the protocol. This type of adversary is used only in HIP because SSL does not incorporate with the client puzzle.

Type 4 adversary (`ad4`) attempts to flood bogus messages at the third step of the protocol by choosing the third message randomly. This type of adversary is used for both SSL and HIP.

To clarify the description of adversaries' ability, the major goal of an adversary type 1 is to overwhelm the server's storage by sending a large number of requested packets, for example, a denial-of-service attack via ping [7] and SYN flooding attack [8], while the major goal of an adversary type 2, 3, and 4 is to force the server to waste computational resources up to the final step of the digital signature verification and digital signature generation which are expensive operations.

During the protocol execution, individual adversary has a specified number of requested tokens at the begining. Moreover, our simulation allows all adversaries to re-generate new bogus messages as soon as they receive returned packets from the responder. That means adversaries have the power to constantly floods new bogus messages to deplete the connection queue. This kind of attack can be considered as ping flooding attacks [7], or TCP SYN flooding and IP spoofing attacks [8]. However, allowing this unlimited ability to adversaries might cause more advantages over honest clients and a responder because adversaries are able to launch such attacks until the responder gets congestion and terminates itself. Therefore, those adversasies are able to deny any services to any websites by keep flooding bogus messages with unlimited power. That might lead to difficult task not only for the defender to protect the communication network from DoS attacks, but also the protocol engineering to develop efficient protocols to resist against such attacks.

In order to model DoS adversaries, one possible way to limit adversaries to gain more advantages and control the system is to specify the attack timing period which can be seen as the time limitation. This approach has an obvious result because the percentage of throughput will be very less during the attack, meanwhile the output becomes the normal level when there is no attack in the network. Another approach is to specify the resource in order to perform the attacks as well as to limit the capacity such as CPU, memory, or bandwidth of adversaries. It means that adversaries must have enough resources available for launching attacks. The later approach seems more interesting and useful to implement than the former one because adversaries are able to perform attacks any times as long as they have available resources.

In our model, adversaries are able to perform the number of attacks depending on the available resources specified at the initial state during the protocol simulation. Before launching the new attacks, adversaries have to wait for a return message (token/available resource) from the responder. In normal situation, the number of returned messages will be equivalent to the number of messages that the responder receives. That means adversaries still have the same level of capability to perform DoS attack as long as the responder can serve those packets. However, once the responder is in a full-loaded condition, the responder starts to reject next arriving messages from any principals that causes adversaries to lose their packets (tokens) from the system. Someone may argue this is not fair to the adversary, however, if we are not limit the DoS attacks ability, the responder is always in a full-load condition and unable to serve any legitimate users, so we are unable to measure the toleration of any key exchange protocols for resisting with DoS attacks.

4 Cost-Based Framework

In this section, we provide a cryptographic benchmark of some specific algorithms. This could be one promising technique used to measure a CPU usage which alternatively be used to represent more specific computational costs instead of an original representation. We can use the total computations for comparing a cost of operations between an initiator and a responder as stated by Meadows. Finally, we present examples of the SSL and HIP cost-based framework.

4.1 Refinement of Meadows's Framework

An obvious limitation of the original formulation of the framework is that the computational costs are not defined precisely, but consist instead of a small number of discrete values. Indeed Meadows herself called this a "crude and ad hoc cost function" [19]. In order to obtain a more useful cost comparison we need to obtain a more accurate estimate of the computational and/or storage costs required to complete the protocol steps. How to do this is not as obvious as it may seem at first.

When comparing efficiency of different cryptographic protocols is it customary to count the number of different types of cryptographic operations. For protocols that use public key operations it is common to ignore most operations and count only the most expensive ones, which typically are exponentiations in different groups (traditionally written as multiplications in elliptic curve computations). However, for the protocols that interest us this is definitely not acceptable. As mentioned above, one common technique in DoS prevention is to demand that clients solve *puzzles* which require the client to engage in some computational effort, such as to iterate a hash function a large number of times. Although one hash computation takes minimal time in comparison with a public key operation, ignoring a large number of hash computations could make the cost function ignore completely the DoS prevention mechanism when a puzzle is used. Therefore we need to be able to compare directly the cost of all different kinds of cryptographic operations.

Comparing operations like hashing and exponentiations directly seems very hard to do since they are based on completely different types of primitive instructions. Therefore we have resorted to an empirical comparison which compares benchmark implementation on common types of processors. While we acknowledge that the detailed results may differ considerably for different computing environments (CPU, compilers, memory, and so on) we believe that the obtained figures are indicative of the true cost in typical environments and allow reasonable comparisons to be made.

For our cost estimates, we use the cryptographic protocol benchmarks of Wei Dai [10]. These include tested speed benchmarks for some of the most commonly used cryptographic algorithms using Crypto++ library[1] version 5.2.1 on a Pentium 4 2.1 GHz processor under Windows XP SP 1. More cryptographic benchmarking has been done by Gupta [13] and Tan [26] on the specific processors; however, they did not test public-key encryption.

Table 2 only presents the results for some specific cryptographic algorithms available for negotiating during the three-way handshake on the SSL protocol and the HIP based exchange defined in HIP specification. The units that we use in Table 2 are kilocycles per block (note that block size varies for different algorithms). This allows direct comparison of CPU usage and may be expected to be similar on processors with different clock speeds. This entails conversion from the original data which uses direct time measurements.

From the table, we are able to estimate the CPU usage in cycles per block for common hash functions and the symmetric key encryption, and cycles per operations for the 1024-bit key lengths of public-key encryption and Diffie-Hellman key exchange algorithm. Once we get a result, we scale it down by a factor of 1000 (kilo) and apply these costs in our formal specification and analysis. Before we can export these values into CPN Tools, we round them into an integer representation because CPN Tools limits only integers in the simulation process.

[1] available at `http://www.eskimo.com/~weidai/cryptlib.html` and `http://sourceforge.net/projects/cryptopp/`

Table 2. Computational Cost of CPU and Time Usage of Specific Algorithms

Hash	kCycle/Block	nsec/bit	Symmetrical Crypto	kCycle/Block	nsec/bit
SHA-1 (512bits/block)	1.89	1.84	**DES** (64bits/block)	0.75	5.86
MD5 (512bits/block)	0.59	0.58	**Blowfish** (64bits/block)	0.25	1.94
HMAC/MD5 (512bits/block)	0.59	0.58	**AES** (128bits/block)	0.53	2.05
Public-Key Crypto	kCycle/ops	nsec/bit	Key Exchange	kCycle/ops	nsec/bit
RSA Encryption/Verification	383.66	187.08	**Diffie-Hellman Key-Pair Generation**	4605.65	2245.80
RSA Decryption/Signature	9985.47	4869.11			
DSA Signature	4569.62	2228.23	**Diffie-Hellman Key Agreement**	8100.69	3950.05
DSA Verification	5239	2554.64			

4.2 Experiment 1: SSL Cost-based Model

Figure 4 shows the simulation of the SSL protocol. As SSL is modeled hierarchically for simplicity of the model and simulation, all nodes in the top page are related to individual subpages defined by the SSL specification. In the top page, it consists of three network segmentations; the initiator (it could be either the honest client who performs as a protocol specification or the adversary who does not play honestly), the responder of the protocol, and the communication network. At this state, we do not permit the adversary to reuse the previous messages to attack the responder, i.e. when the adversary attempts to flood new bogus messages, the adversary has to participate with the construction by computing individual messages.

Because SSL protocol is modeled in the cost-based framework, every single state has been attached with the computational cost-place for displaying the operational effort of that state during the protocol execution. During the protocol execution, we record all operational costs of individual transitions by adding a CPU usage data from Table 2.

At the begining, there are three types of user who can request for services; the honest client (`hc`), the adversary type one (`ad1`) who attempts to attack the protocol by flooding initial request messages, and the adversary type four (`ad4`) who attempts to attack the protocol at the third messages, indicated in the user token. Morever, we are able to specify the number of messages sent simultaneously from the initiator, define the string of messages, as well as investigate the computational cost[2] when the message travels to each operation in this token.

To examine our cost-based model, the initiator starts sending a request message to the responder. At the initial phase, there is no operation to make a cost to the message. Once the responder receives a request message, the responder has to choose the Diffie-Hellman (DH) parameter used for generating a session key and returns with the certificate in the second message. At this step, the responder has to store the received information and open the connection until the responder receives the replied message from the initiator because SSL session is the stateful protocol. This condition subjects to a flooding attack that presents a risk

[2] It is important to note that the display cost at each state shows the total operation cost of that corresponding state only, not an accumulation cost of all state. The reason is that it is easy to compare the cost of message construction on the initiator's machine with the cost of protocol engagement on the responder's machine at every single step of the protocol as suggested by Meadows [19].

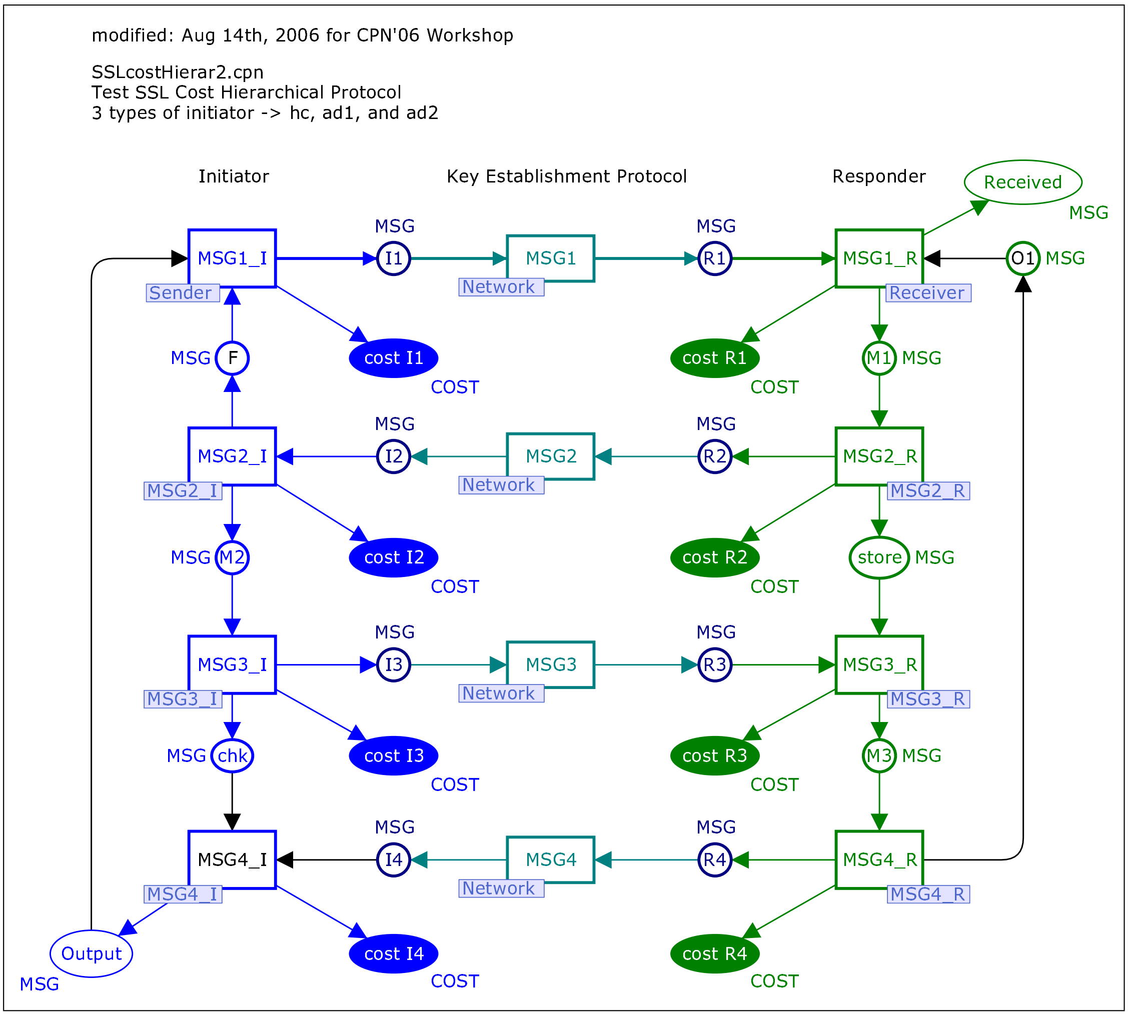


Fig. 4. SSL Cost-based Construction

to the responder to exhaust its connection queues. The stored information during the protocol run is shown in the store-place under the MSG2_R position in the top page (figure 4).

Upon receipt of a reply message, the initiator has to select a *DH* value for calculating a session key used to protect the communication. The initiator also requires to verify the responder's certificate and subsequently to extracts the public key for encrypting the *DH* value. At this phase, the adversary might send a large number of bogus messages to the responder to exhaust the responder's computational resources used for verifying the initiator's identity if the client authentication mode is selected. If no client authentication is required by the responder, Type 4 adversary is able to choose message 3 randomly for depleting the responder's resource more easily. That is because the responder has to waste its resources to decrypt message 3 which is encrypted using the public-key cryptosystem (the RSA decryption algorithm takes more magnitude of CPU usage than RSA verification).

In the final step, the responder obtains the session key by using its private key to decrypt the initiator's message. If this process is success, the responder uses this session key to produce hashed-MAC (HMAC) and returns it to the initiator for a key confirmation. Once the initiator receives this message, the initiator decrypts it to check the correctness of the key.

Authentication Protocol	**Initiator**		**Responder**	
			with Client Authentication	without Client Authentication
SSL	***hc***	5376	14978	14594
	ad1	0	4606	4606
	ad4	0	4990	14592

Table 3. Comparison of the SSL Computational Cost with and without Client Authentication

As described above, SSL has two vulnerabilities to the DoS attacks at two states, following message 1 and message 3. The first vulnerability is demonstrated in Figure 4 at the store position (at the middle right under the transition MSG2_R). In the second vulnerability, the analyst can see the computational cost from Table 3 when comparing the cost of computation between the initiator and the responder. The total cost of the responder is greater than the initiator because the responder has to participate with the RSA decryption, which is an expensive operation, in the third phase. Meanwhile the adversary does not spend resources to verify the second message in the second step and employs only cheap computations (because we have not defined the cost of the adversary to reuse, spoof, insert, or interrupt the message) to send the third message to deny services.

To sum up, as SSL is designed with message efficiency rather than resistance to DoS attacks in mind, these situations reveal the denial-of-service threats to the responder. Recently, there are several well-known techniques used as a denial-of-service tool to attack the communication running over the SSL protocol. One example is a SYN flooding attack [8] in which the adversary requires only little computational effort for constructing bogus messages to deplete the responder's resource, while the responder requires greater magnitude compared to the adversary for handling these messages.

4.3 Experiment 2: HIP Cost-based Model

The purpose of this simulation is to compare computational cost of the protocol execution between all principals with some possible ranges of puzzle difficulty (k) including $k = 0$ (which means that no puzzle is required), easiest value $k = 1$ for contrasting the difference between `ad3` and `ad4`, intermediate values $k = 10$, $k = 20$, and $k = 40$ for a hardest value as instructed by the designer. Similar to the SSL-model, we insert a cost-place to individual transition for displaying a computational cost of every single step. A measurement of CPU usage has been used to indicate individual steps and compared the total cost among an initiator, individual adversary type, and a responder as a key concept of cost-based analysis specified by Meadows for comparing cost of protocol engagement of individual principal.

In this simulation, we allow individual initiator to initiate a request token only once, while the responder is able to flexibly adjust the puzzle difficulty within defined values. Once the simulation has arrived to the final step, we record a total computational cost of individual user comparing to the responder on specified ranges of k. The HIP cost-based model is demonstrated in Figure 5.

During the protocol execution, the initiator sends a request message to the responder using the host identity tag (HIT) which is a hash of the host identifier (HI) used in HIP payload and to index the corresponding state in the end hosts. Therefore, the initiator only employs cheap operations at the beginning step. We assume that the computation at this step can be precomputed, so, the cost at the first operation would be negligible. Once the responder receives the requested message, the responder requires a hash operation and some values from the precomputation for returning to the initiator in the second step. This operation is estimated as a cheap operation similar to the initiator.

When the initiator receives the replied message in the MSG2_I subpage, the initiator first verifies the HIT and responder's signature. There are three possible outputs after verification depending on the user field; 1) if the user is `hc`, the token will traverse to accept-place and the cost is equal to the HIT verification plus

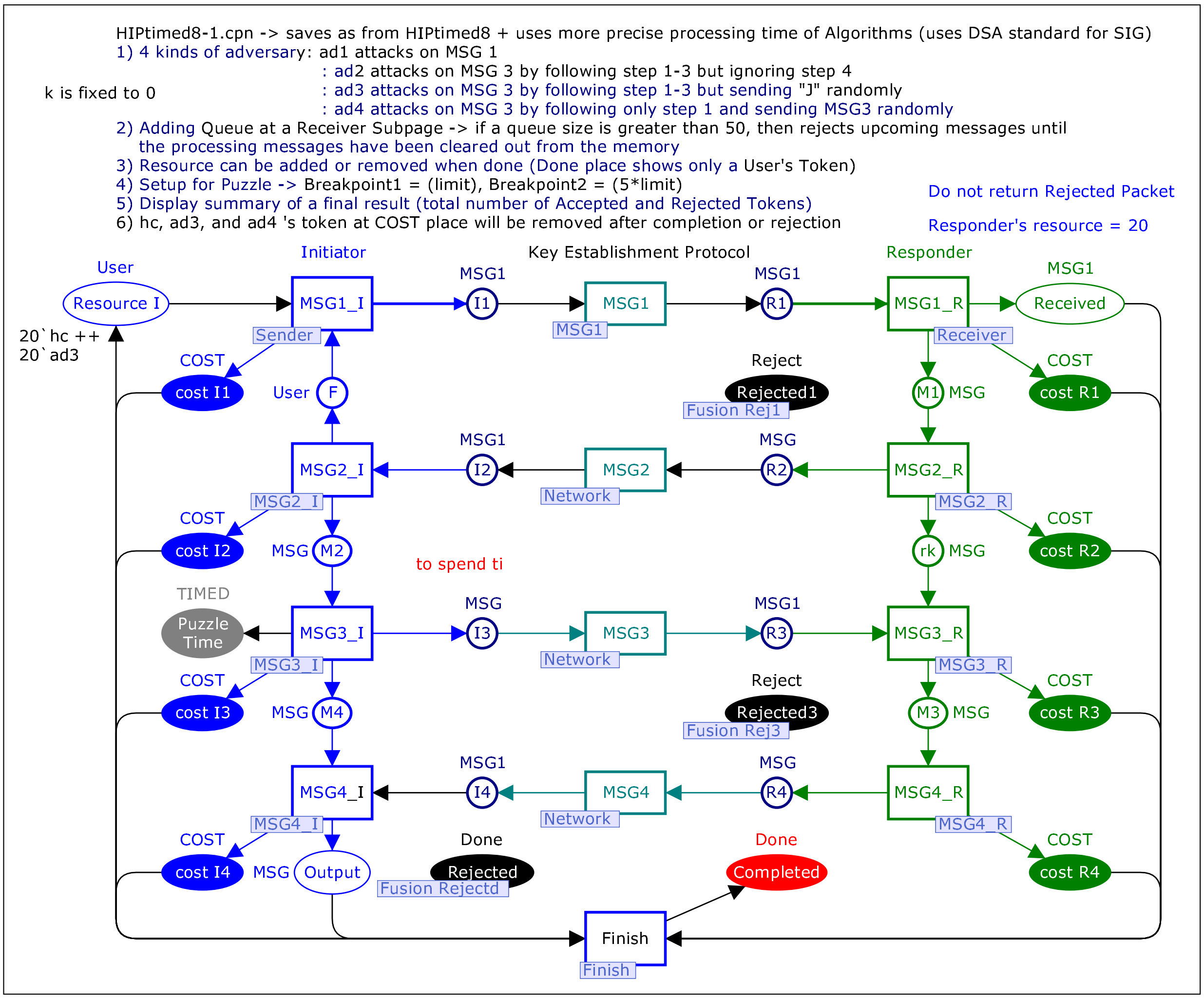

HIPtimed8-1.cpn -> saves as from HIPtimed8 + uses more precise processing time of Algorithms (uses DSA standard for SIG)
1) 4 kinds of adversary: ad1 attacks on MSG 1
: ad2 attacks on MSG 3 by following step 1-3 but ignoring step 4
: ad3 attacks on MSG 3 by following step 1-3 but sending "J" randomly
: ad4 attacks on MSG 3 by following only step 1 and sending MSG3 randomly
k is fixed to 0
2) Adding Queue at a Receiver Subpage -> if a queue size is greater than 50, then rejects upcoming messages until the processing messages have been cleared out from the memory
3) Resource can be added or removed when done (Done place shows only a User's Token)
4) Setup for Puzzle -> Breakpoint1 = (limit), Breakpoint2 = (5*limit)
5) Display summary of a final result (total number of Accepted and Rejected Tokens)
6) hc, ad3, and ad4 's token at COST place will be removed after completion or rejection
Do not return Rejected Packet
Responder's resource = 20
User
Initiator
Key Establishment Protocol
Responder
Resource I
20`hc ++
20`ad3
MSG1_I
Sender
MSG1
I1
MSG1
MSG1
MSG1
R1
MSG1_R
Receiver
MSG1
Received
COST
cost I1
User
F
Reject
Rejected1
Fusion Rej1
M1
MSG
COST
cost R1
MSG2_I
MSG2_I
MSG1
I2
MSG2
Network
MSG
R2
MSG2_R
MSG2_R
COST
cost I2
MSG
M2
to spend ti
rk
MSG
COST
cost R2
TIMED
Puzzle Time
MSG3_I
MSG3_I
MSG
I3
MSG3
Network
MSG1
R3
MSG3_R
MSG3_R
COST
cost I3
MSG
M4
Reject
Rejected3
Fusion Rej3
M3
MSG
COST
cost R3
MSG4_I
MSG4_I
MSG1
I4
MSG4
Network
MSG
R4
MSG4_R
MSG4_R
COST
cost I4
MSG
Output
Done
Rejected
Fusion Rejectd
Done
Completed
COST
cost R4
Finish
Finish

Fig. 5. HIP Cost-based Construction

signature verification, 2) if the user is ad2, ad3, and ad4, the token will also traverse to accept-place but the cost is only zero because it performs nothing at this step, and 3) if the user is ad1, the token will traverse to fail-place because ad1 does not take any further actions after the first message has been sent, therefore, the computational cost of ad1 is zero for the second stage.

The operations in message three of the initiator include the brute-force search to find the puzzle solution, and the key generation. The cost of solving a puzzle depends on the value of k including $k = 0$, $k = 1$, $k = 10$, $k = 20$, and $k = 40$ in the puzzle message field. However, hc, ad2, and ad3 are required to solve the puzzle solution. Like ad1, the ad4 does not attempt to solve the puzzle, as a result, the puzzle difficulty does not affect to the computational cost on this type of adversary. Another important thing to note is that, the cost of the adversary to spoof, insert, or interrupt the message has not been defined in this phase. So, we set the cost of randomly chosen messages in the case of ad3 and ad4 to be zero.

Considering the responder's task, when the responder receives the third-step message from the initiator, the responder begins to validate the puzzle solution which is defined as a cheap authentication process because the responder performs only one hash calculation. If it is invalid, the process will stop and the responder will drop the corresponding packet from the connection queue (the system will return a resource to the responder). Otherwise, the responder performs the decryption to obtain an initiator's public key.

The responder finally verifies the signature by using the initiator's public key obtained in the previous step. The result would be either valid or invalid. After the authentication has been completed, the responder and the initiator will perform a key confirmation and start to exchange information. Table 4 summarizes the computational cost when the puzzle difficulty is set to k=1 or k=10 comparing between every principal and the responder. The experimental result shows that the most effective adversary is `ad3` (the greatest different threshold between `ad3` and the responder) because `ad3` can force the responder to engage in the expensive tasks, i.e. digital signature verification.

Table 4. Comparison of Computational Cost of HIP with k=1 and k=10

Authentication	**Initiator**			**Responder**		
Protocol		$\mathbf{k}$**=1**	$\mathbf{k}$**=10**	J,$E1$,sig_I valid	only J valid	Everything invalid
HIP	***hc***	19973	22017	19591	-	-
	ad1	0	0	-	-	2
	ad2	14982	17026	19591	-	-
	ad3	4	2048	-	4998	-
	ad4	0	0	-	-	6

Figures 6 illustrate the computational cost of the honest client, Type 2, and Type 3 adversary, respectively. In that comparison charts, we measure the cost of those users who involve with solving the puzzle of the difficulty level $k = 0, 1, 10, 20, 40$.

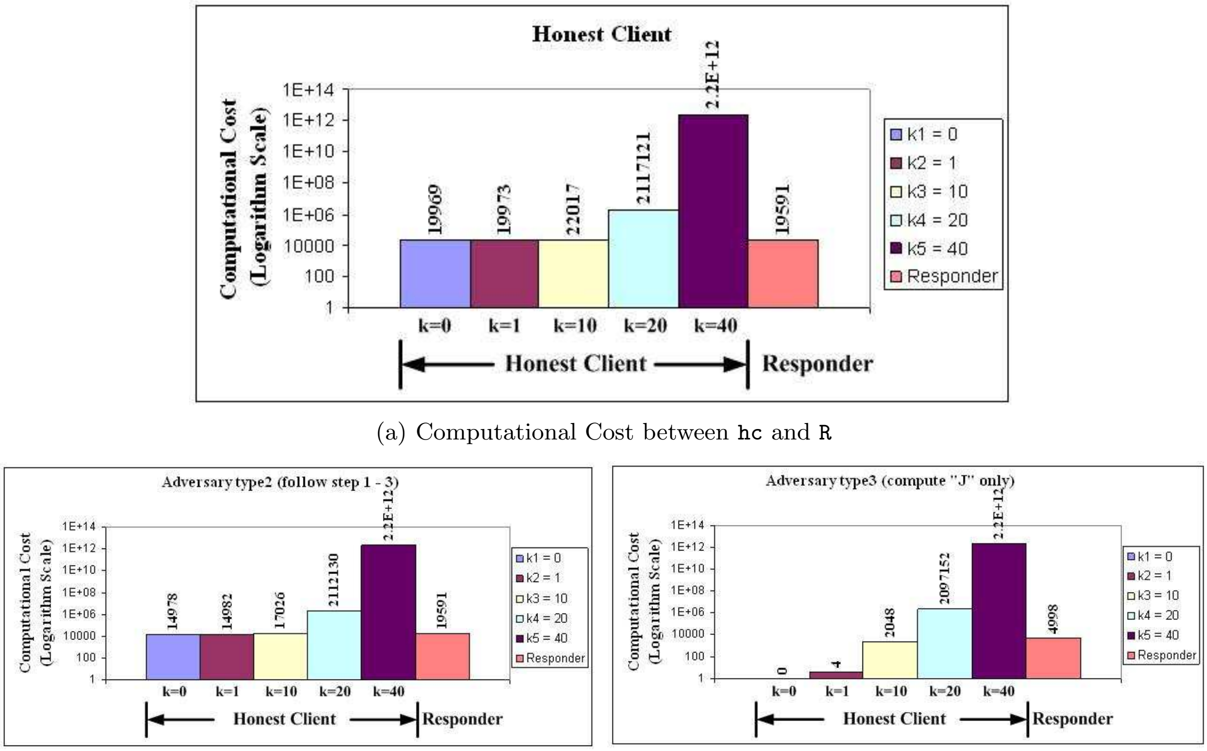
Comparison between Figures 6(a) and 6(b) shows that `hc` and `ad2` incur similar computational costs for the same value of k chosen. This illustrates well the effectiveness of HIP in achieving its aims in resisting DoS attacks, at least against this type of adversary. On the other hand, `ad3` and `ad4` spend very small computational resources compared with the responder because both adversaries use some random message elements. This situation would bring the responder to the risk of DoS attacks if the value of k is not chosen properly. Figure 6(c) indicates that a value of k a little above 10 would be appropriate to ensure that `ad3` uses as much computation as the responder.

5 Timed Coloured Petri Nets (Timed CPNs)

We attempt to design cryptographic protocols more realistic by adopting the concept of Timed Petri Nets into our implementation, i.e all cryptographic processes require some amount of time calculated by using cryptographic benchmark of Crypto++ library developed by Wei Dai [10]. In the Timed Petri Nets, the concept of *the simulated time* or *the model time*[3], which is represented by the system clock in the tool, has been introduced. Once we have attached the system time into tokens, we can see the sequence or action of states that tokens move as a temporal analysis, that means only tokens which hold the current time as presenting on the clock can be moved to the next step, while the others have to wait until the system clock reaches their value.

To develop a model on CPN Tools, HIP is constructed as a hierarchical construction for simplicity of the model and simulation. All nodes in the top page are related to individual subpages defined by the HIP specification [21]. HIP is modeled in the cost-based framework, so each state has the computational cost place to display the total cost of that state during the protocol simulation. Furthermore, the concept of the responder's resource is used in this evaluation. Once the responder has to deal with requests, the responder

[3] More formal descriptions are available on the official website of CPN Tools, `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`



(a) Computational Cost between `hc` and `R`

(b) Computational Cost between `ad2` and `R`

(c) Computational Cost between `ad3` and `R`

Fig. 6. Comparison of Computational Cost on HIP with different ranges of k

spend one resource for individual request. It means that if incoming packets beyond the responder capacity, the responder then rejects the further incoming packets until he has either done the legitimate traffics or detected bogus messages and removed them from the storage.

In our model, we initially configure an individual message to contain four coloured sets; 1) *User* who initiates the token, 2) *NUM* which is the number of messages sent simultaneously from the initiator, 3) *DATA* which indicates the string of messages, and 4) *COST* which is used to display the computational cost when the message is traveling to each operation. Similar to other models, the display cost at each state shows the total operation cost of that corresponding state only, not an accumulation cost of all state. The top page of HIP Timed-CPNs is constructed as demonstrated in Figure 5.

From Figure 5, the top page consists of three major segments; 1) an initiator's network, 2) a communication channel, and 3) a responder's network. Each transition represents the stage of protocol execution, which consists of four stages in each principal because HIP is a four exchanging messages protocol, corresponding to specified subpage. In each stage, it consists of CPN elements constructed as specified in HIP protocol specification [21]. An example of responder's subpage at the first stage is demonstrated in Figure 7

This responder subpage consists of two main important transitions which are used for arranging the order of requested messages and verifying the validity of the responder's host identity tag (HIT_R). Considering the *Queue* and *Count* transition, the purpose of these transitions are to measure the number of arriving requested packets in order for the responder to flexibly and appropriately adjust the puzzle difficulty associating to the number of workloads. Moreover, in the situation when multiple requested messages arriving to this transition simultaneously, the process of arranging the order of such packets is random based on CPN Tools.

Another transition in this subpage is $CheckHIT_R$. In order to process the job, the responder has to have available memory resource. Otherwise, the responder will reject requested messages until some of them are removed from the connection queue. In order to define the responder's capability, we insert a *resource* place

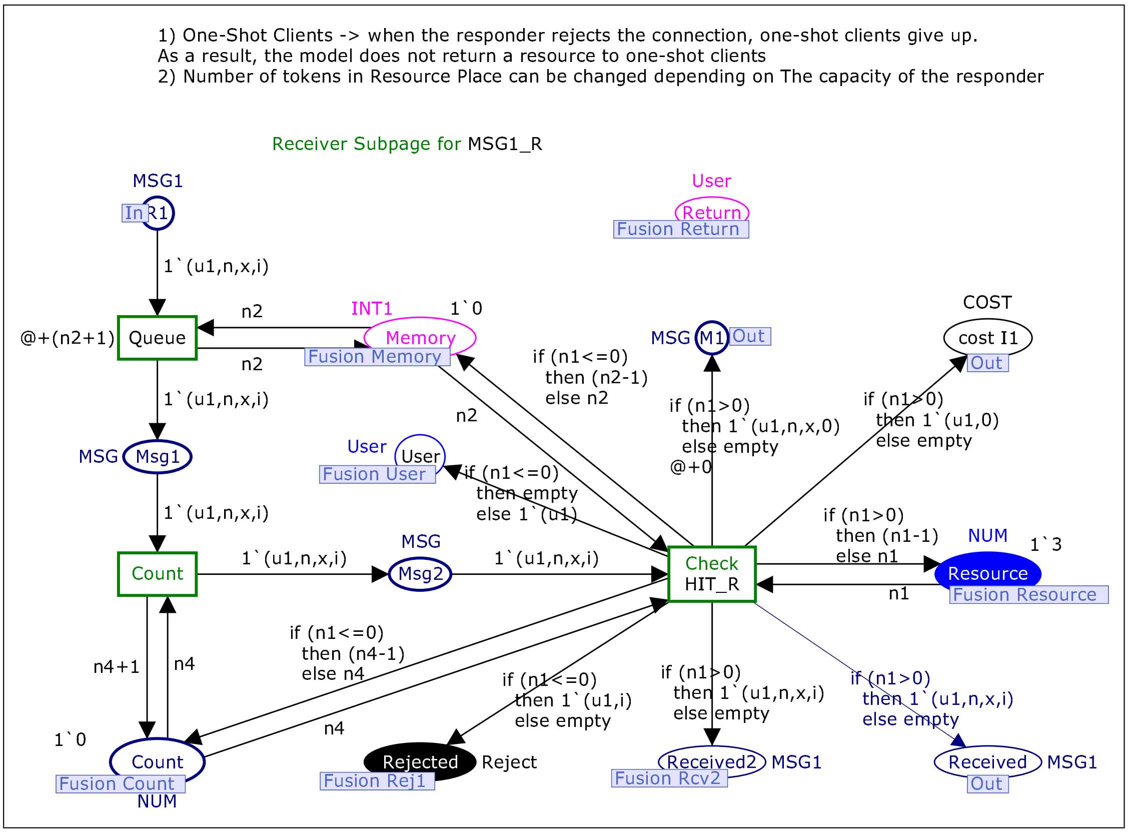


Fig. 7. The Responder's Subpage at the First Stage

in this stage. We have to initially specify the number of available connection queue before activating the simulation. This number represents the responder's capacity (in the case of *memory* resources) in order to serve initiator's messages simultaneously without degradation of services. Note that, each packet requires only one resource (one connection queue) during the process at the responder's machine. Once those messages are done and removed from the connection queue, they will return resource token to the responder's machine.

In addition, the quantity at the resource place represent not only for available connection queue, but indirectly represent *CPU* usage on the responder's machine as well. We shall explain this concept by giving an example. Comparing two messages in which message one is in the stage one, while message two is in the stage three. The responder has to spend similar amount of connection queue, a token per message, for serving both of them. However, in the case of CPU usage, the responder has to waste more power for message two than message one because main tasks at stage three is to verify the puzzle solution and the signature, while the task at stage one is only choosing the puzzle difficulty and return it to the initiator. By specifying the time usage from Table 2 for individual cryptographic transition, we can imply that the longer period that the message is processed in the responder's machine, the more CPU usage does it take from the responder.

5.1 Experiment 3: Non-adjustable Client Puzzles

The purpose of the third experiment is to examine the minimal DoS-resistant mechanism. To achieve this, we run the simulation under four specified attacks and the combination of four strategies (defined as *All*) with the non-adjustable client puzzle. We initially fix k=1, i.e. the easiest value[4], because hc prefers to spend nothing expensive for establishing a connection under normal circumstances.

Additional from the experiment of a HIP cost-based model, we allow a responder to participate with a pair of initiator (hc and an individual type of ad). We assume that the responder has to deal with different strategies of adversary and different amount of packets which consist of both legitimate and bogus messages. Considering to a number of packet, hc can initiate the amount of requests (C) at 80%, 100%, and 150% of the responder's capacity (R). Meanwhile, a single type of ad can flood the amount of bogus requests (Z) at 100%, 200%, and 1000% of the responder's capacity (R).

In order to examine the tolerance of HIP protocol under different attack strategies, individual adversary has been made a pair with an honest client during the protocol execution. Under four specified attacks, there are three possible situations to cause a responder to waste computational resources by the adversary.

1. All values of the third message including a puzzle solution J, an encrypted part $K_e\{HI_I\}$, and a digital signature sig_I are valid. This will force the responder to process a gradual authentication and complete the final step of the communication.
2. Only the client puzzle solution J is valid. This situation also causes the responder to perform the puzzle solution verification, decryption, and the signature verification. The responder can detect the attack only at the final step of authentication.
3. The client puzzle solution J is invalid, so the responder computes only a cheap hash function to verify the solution and then this connection will be terminated whether the remaining messages are valid.

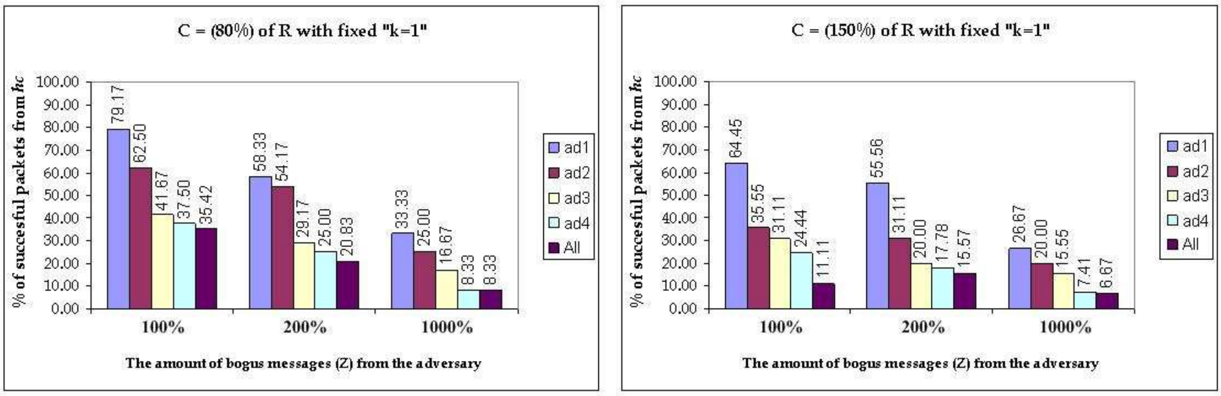
Finally, by inserting places for displaying the number of completed and rejected messages at the responder's network, the number of successful legitimate requests that the responder can serve under different adversary's abilities are measured as the percentage for the protocol evaluation.

The Experimental Results: Figure 8 represents the percentage of successful legitimate connections compared among three different amount of bogus messages (Z=100%, 200%, and 1000% of the responder's capacity R) from five adversarial strategies (in the combination strategy, *All*, each of adversary type has the same amount of bogus messages that makes the total number equivalent to the specified quantity). When we prohibit the responder's ability to adjust k, the percentage of successful messages from hc to obtain a service will drop drastically when ad increases the number of bogus messages. Comparing different types of ad, the most effective is ad4 who sends bogus messages to the responder by crafting messages randomly. This is because ad4 can flood a large number of messages to overwhelm the responder's resource quicker than the others, which causes the responder to reject the next incoming messages from hc. Although packets from ad will be detected and discarded by the responder, these packets can be re-generated and flooded by ad as soon as ad receives returned packets from the responder at phase two.

Comparing ad1 and ad4, even though both of them craft random messages, ad4 can achieve the goal at higher rate than ad1 because the responder can process the incoming request at step 1 and clear a queue faster than at step 3. At step 1, the responder only participates in the protocol by choosing the puzzle difficulty (k) and pre-computed information, and returns it to ad1. Although, ad1 can re-generate bogus messages after receiving replied messages, this does not cause the responder to reject a large number of messages because HIP mitigates such problem by adopting a stateless-connection. On the other hand, the task of ad4, to fill-up the responder's queue at step 3, can be achieved more easily than ad1 because the process of checking a puzzle solution and a digital signature takes longer than a whole process at step 1.

Considering ad2 and ad3 who attempt to deny service at phase 3 by computing the puzzle solution, the results show that ad3 succeeds at higher proportion than ad2. This is because ad3 can flood attack messages faster than ad2 who must engage in the correct generation of message two. Nonetheless, both adversaries can force the responder to engage in the signature verification. Although ad4 can flood large number of messages at step 3 as well as ad2 and ad3, ad4 cannot force the responder to engage in expensive operations because the responder is able to detect the message forgery at the cheap puzzle verification process. However,

[4] If we choose k=0 which means no client puzzle is required, we cannot see the difference of costs between ad3 and ad4 because the task of both adversaries will be the same.



(a) `hc`'s load = 80% of `R`

(b) `hc`'s load = 150% of `R`

Fig. 8. Percentage of throughput from `hc` with k=1

without the assistance of puzzle difficulty, the percentage of successful messages in the case of `hc` and `ad4` is lower than the others because `ad4` floods message three at the highest rate. As a result, the most effective adversary to deny services on the responder would be `ad4` that attacks the verification phase. Most key agreement protocols incorporate verification tasks that would be susceptible to resource exhaustion attacks.

Finally, the result of the combination of all attack techniques shows that when the responder has to deal with all types of adversary, the percentage of legitimate users served by the responder will fall significantly with increment of bogus messages. Once we can identify the most effective scenario, we will apply this technique to the fourth experiment for investigating the usefulness of puzzle difficulty.

5.2 Experiment 4: Adjustable Client Puzzles

The purpose of this experiment is to measure a toleration of the responder when adjustable client puzzles are implemented. The result can be used to compare with the experiment on the cost-based model for comfirming that whether Meadows's cost-based framework is efficient for evaluating the DoS-resistant protocols or not.

To examine the protocol, we allocate two possible values, $k = 1$ and $k = 10$ for the responder. We choose those two values because they do not put more computational effort to `hc` and the total amount of task is still in the acceptable threshold comparing to tasks on the responder (see Figure 6). In order to allow the responder to flexibly adjust puzzle difficulty between those two values more efficiently, we simply insert the counter into the model for measuring the condition of a responder's workload. Once the workload has reached the maximum tolerance, the responder will increase the puzzle difficulty to the higher level for delaying the incoming rate of requested messages.

At the beginning of the protocol assessment, we allow both `hc` and an individual type of `ad` to make requests at the same time to the responder. However, the responder is able to process requests only one message at a time. So, a concept of the responder queue is implemented for arranging an order of incoming packets. Figure 9 illustrates a construction and a simulation of the HIP protocol by means of Timed CPNs.

Considering the initiator's packet, there are different actions from initiators and the responder during the protocol run. The `hc` initiates a request only once and keep waiting to process next steps. This delay is described by means of Timed CPNs, i.e. every transitions which relates to cryptographic operations is defined as a timed process. During the simulation, if requests from `hc` have been rejected under DoS circumstances, `hc` gives up to open another session. On the other hand, there are two different situations that packets from `ad` are rejected by the responder; 1) the responder detects the bogus messages during the verification steps, and 2) the responder does not have enough resources for serving any requests. Once the responder detects the attack and rejects those packets, `ad` will lose those packets from the system.

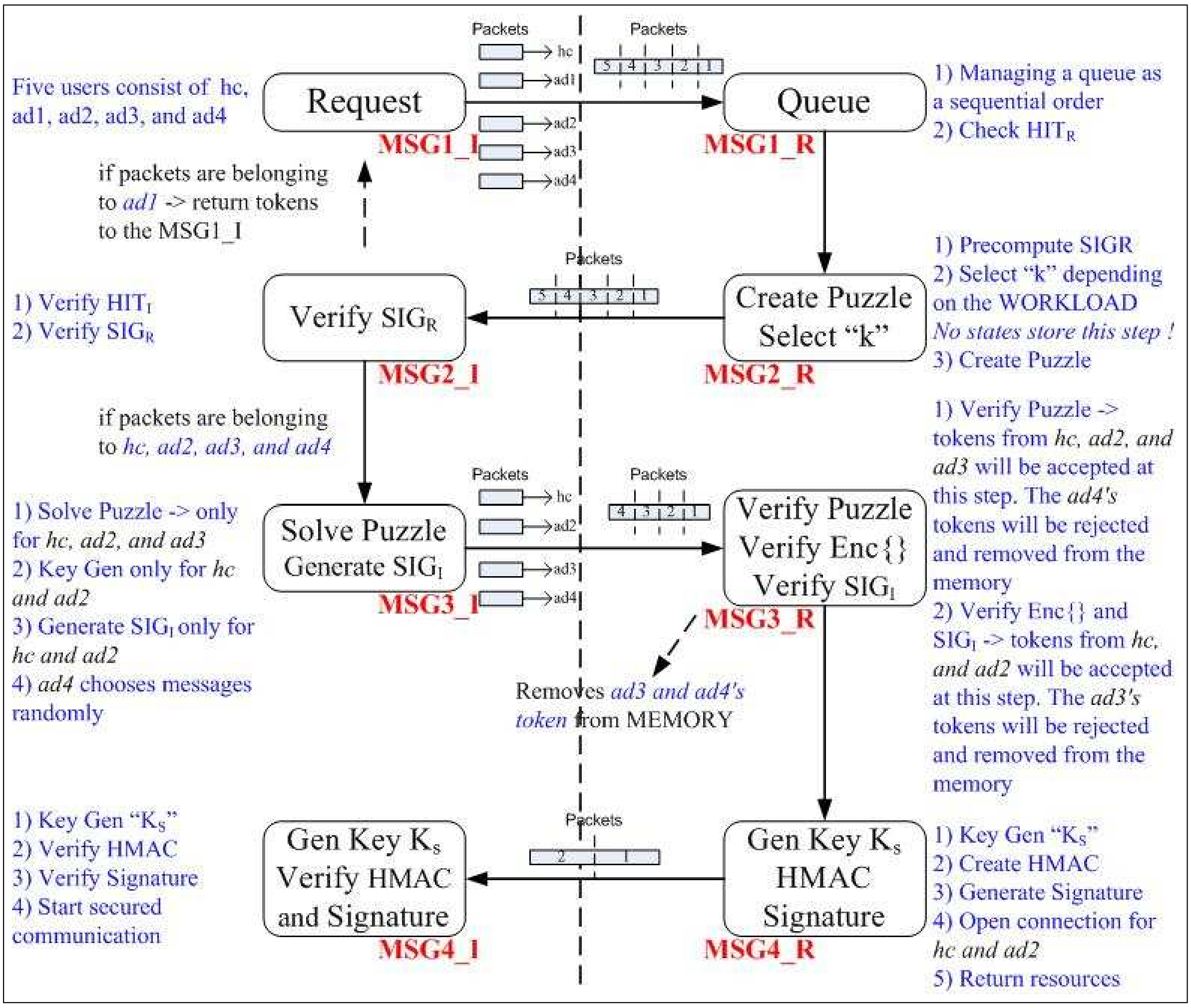
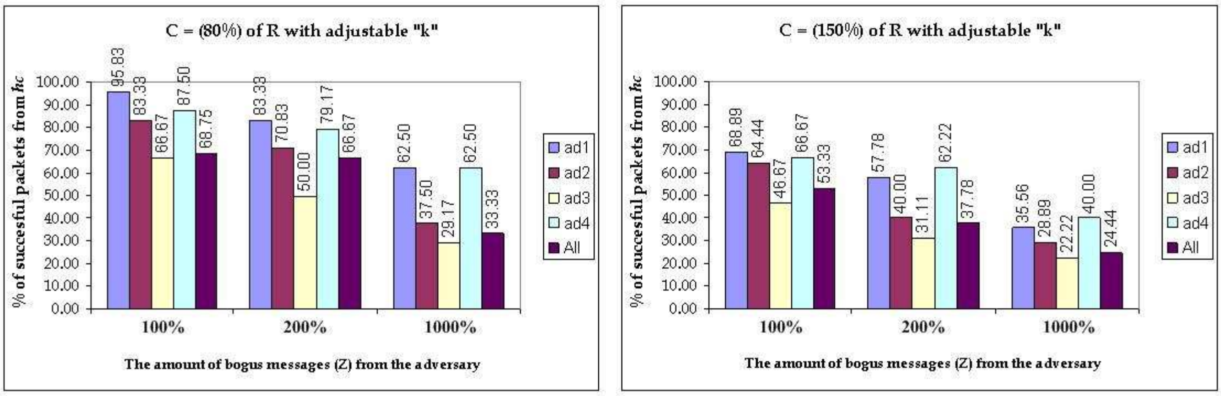

Five users consist of hc, ad1, ad2, ad3, and ad4
Request
MSG1_I
Packets
hc
ad1
ad2
ad3
ad4
Packets
5 4 3 2 1
Queue
MSG1_R
1) Managing a queue as a sequential order
2) Check HIT_R
if packets are belonging to ad1 -> return tokens to the MSG1_I
Packets
5 4 3 2 1
Verify SIG_R
MSG2_I
Create Puzzle
Select "k"
MSG2_R
1) Verify HIT_I
2) Verify SIG_R
1) Precompute SIGR
2) Select "k" depending on the WORKLOAD
No states store this step !
3) Create Puzzle
if packets are belonging to hc, ad2, ad3, and ad4
1) Solve Puzzle -> only for hc, ad2, and ad3
2) Key Gen only for hc and ad2
3) Generate SIG_I only for hc and ad2
4) ad4 chooses messages randomly
Solve Puzzle
Generate SIG_I
MSG3_I
Packets
hc
ad2
ad3
ad4
Packets
4 3 2 1
Verify Puzzle
Verify Enc{}
Verify SIG_I
MSG3_R
1) Verify Puzzle -> tokens from hc, ad2, and ad3 will be accepted at this step. The ad4's tokens will be rejected and removed from the memory
2) Verify Enc{} and SIG_I -> tokens from hc, and ad2 will be accepted at this step. The ad3's tokens will be rejected and removed from the memory
Removes ad3 and ad4's token from MEMORY
1) Key Gen "K_S"
2) Verify HMAC
3) Verify Signature
4) Start secured communication
Gen Key K_s
Verify HMAC
and Signature
MSG4_I
Packets
2 1
Gen Key K_s
HMAC
Signature
MSG4_R
1) Key Gen "K_S"
2) Create HMAC
3) Generate Signature
4) Open connection for hc and ad2
5) Return resources

Fig. 9. Protocol Steps of HIP Timed CPNs Model

The Experimental Results: To adjust the puzzle difficulty, we allocate two possible values for the responder to determine. Under normal circumstances, the responder selects k=1, which means the easiest puzzle solution is required from the initiator. Once the responder receives more requested packets than its maximum capacity to handle, the responder raises the puzzle difficulty. In the experiments described here, we choose k=10. Because this puzzle technique is a hash-based puzzle, this value will help the responder to slow down the incoming rate by requiring the work of the initiator to solve a puzzles at the factor of 2^{10}.

Similarly to the representation of Figure 8, Figure 10 illustrates that the number of attacking machines that the responder can tolerate is increased to a higher proportion compared to the result of experiment 1. Another interesting result is that the successful rate of an honest client's message in the case of ad4 is higher than for the fixed value k=1. The reason is that ad4 does not compute the puzzle solution, so, no matter what the puzzle difficulty is, ad4 can flood the bogus messages at the similar speed as experiment 1. However, at that amount of bogus messages, there are only messages from ad4 (no legitimate traffic because hc has to spend some amount of time to solve the puzzle solution), or just a few messages from hc that arrive to the connection queue before the responder increases puzzle difficulty. As a result, the responder can validate the puzzle solution before the next group of messages has arrived. Undoubtedly, these bogus messages from ad4 will be rejected at the first step of verification which requires only short period and removes such attack



(a) `hc`'s load = 80% of `R`

(b) `hc`'s load = 150% of `R`

Fig. 10. Percentage of throughput from `hc` with k is chosen between 1 and 10

from the connection queue. However, this situation does not occur in the case of `ad3` because they have to spend some amount of time to solve the puzzle as well as `hc`.

6 Conclusion and Future Work

This work has achieved the aims of extending the Meadows's cost-based framework to provide more accurate representation of computational cost and shown the potential of automated analysis. Moreover, we have explored unbalanced computational vulnerabilities on HIP which cause the responder to deplete resources and then terminate all processes by developing formal analysis based on Meadows's cost-based framework and Time CPNs simulation-based analysis. By comparing experimental results from both techniques, we have found a limitation of Meadows framework in order to define the ability of advanced adversaries and address DoS vulnerabilities in DoS-resistant protocols.

In future work, we plan to extend this research by using the model checking capabilities of CPN tools to automatically verify the system by traversing the model and checking whether the cost tolerance between initiator and responder exceeds some reasonable threshold. Moreover, the power of adversaries can be extended in different ways in order to model more powerful attacks. For example, the advanced adversary, who attemps to attack the protocol at the third message, can be extended to flood reused packets from previous connections, eavesdrop messages from a valid communication, or craft bogus messages using existing messages including valid or invalid puzzle solutions as well as digital signatures. By inserting such advanced abilities to the model, we also require a technique to measure and identify cost of those operations in order to achieve a formal analysis.

In addition there are a number of other promising directions for this research.

- Our model can be used to analyse a variety of protocols to provide a comparison of the effectiveness of different protocols in DoS prevention.
- Our model can be integrated into a model for security analysis of authentication and key establishment properties to create a unified protocol analysis tool covering resistance to DoS attacks as well as more traditional security goals.

References

1. W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *the 9th ACM conference on Computer and communications security*, pages 48–58, Washington, DC, USA, 2002. ACM Press.

2. I. Al-azzoni. The Verification of Cryptographic Protocols using Coloured Petri Nets. Master of Applied Sciences Thesis, Department of Software Engineering, McMaster University, Ontario, Canada, 2004.
3. T. Aura, A. Nagarajan, and A. Gurtov. Analysis of the HIP Base Exchange Protocol. In *Proceedings of 10th Australasian Conference on Information Security and Privacy (ACISP 2005)*, volume 3574 of *Lecture Notes in Computer Science*, pages 481 – 493, Brisbane, Australia, Jun 2005. Springer-Verlag.
4. T. Aura and P. Nikander. Stateless Connections. In *International Conference on Information and Communications Security*, pages 87–97, Beijing, China, Nov 1997. Springer-Verlag.
5. T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Security Protocols Workshop 2000*, pages 170–181. Cambridge, Apr 2000.
6. J. Beal and T. Shepard. Deamplification of DoS Attacks via Puzzles. Available: `http://web.mit.edu/jakebeal/www/Unpublished/puzzle.pdf`, 2004.
7. Computer Emergency Response Team (CERT). Denial-of-Service Attack via ping. [Online]. Available: `http://www.cert.org/advisories/CA-1996-26.html` [Accessed: August 2004], 1996.
8. Computer Emergency Response Team (CERT). TCP SYN Flooding and IP Spoofing Attacks. [Online]. Available: `http://www.cert.org/advisories/CA-1996-21.html`, 1996.
9. S. Christensen and K. H. Mortensen. Teaching Coloured Petri Nets- A Gentle Introduction to Formal Methods in a Distributed Systems Course. In *ICATPN*, pages 290–309, 1997.
10. Wei Dai. Crypto++ 5.2.1 Benchmarks. [Online]. Available: `http://www.eskimo.com/~weidai/benchmarks.html` [Accessed: Nov 2005], 2004.
11. E. M. Doyle. Automated Security Analysis of Cryptographic Protocols using Coloured Petri Net Specification. Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada, 1996.
12. A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0. Internet Draft, Internet Engineering Task Force, November 1996. `http://wp.netscape.com/eng/ssl3/draft302.txt`.
13. V. Gupta, S. Gupta, S. Chang, and D. Stebila. Performance Analysis of Elliptic Curve Cryptography for SSL. In *WISE'02: Proceedings of the 3rd ACM Workshop on Wireless Security*, pages 87–94, Atlanta, GA, USA, 2002. ACM Press.
14. Y. Han. Automated Security Analysis of Internet Protocols using Coloured Petri Net Specification. Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada, 1996.
15. M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS 99)*, Sep 1999. Also available as http://citeseer.nj.nec.com/238810.html.
16. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use.* Springer-Verlag, 2nd edition, Vol. 1-3, April, 1997.
17. A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In *the 1999 Network and Distributed System Security Symposium (NDSS '99)*, pages 151–165, San Diego, California, USA, Feb 1999. Internet Society Press, Reston.
18. P. Karn and W. A. Simpson. Photuris: Session-Key Management Protocol. Experimental RFC 2522, IETF, Mar 1999. `http://www.ietf.org/rfc/rfc2522.txt`.
19. C. Meadows. A Cost-Based Framework for Analysis of DoS in Networks. *Journal of Computer Security*, 9(1/2):143–164, Jan 2001.
20. H. C. Moon. A study on formal specification and analysis of cryptographic protocols using Colored Petri Nets. Master of Science Thesis, Institute of Science and Technology, Kwangju University, Korea, 1998.
21. R. Moskowitz. The Host Identity Protocol (HIP). Internet Draft, Internet Engineering Task Force, Jun 2006. `http://www.ietf.org/internet-drafts/draft-ietf-hip-base-06.txt`.
22. B. B. Neih and S. E. Tavares. Modelling and Analysis of Cryptographic Protocols using Petri Nets. In *Advances in Cryptology*, pages 275–295, Berlin, German, 1993.
23. C. A. Petri. *Kommunikation mit Automaten.* PhD Thesis, Institut fur Instumentelle Mathematik, Schriffen des IIM, 1962.
24. J. Smith, J. M. González Nieto, and C. Boyd. Modelling Denial of Service Attacks on JFK with Meadows's Cost-Based Framework. In *Fourth Australasian Information Security Workshop (AISW-NetSec 2006)*, volume 54, pages 125–134. CRPIT series, 2006.
25. J. Smith, S. Tritilanunt, C. Boyd, J. M. González Nieto, and E. Foo. DoS-Resistance in Key Exchange. *International Journal of Wireless and Mobile Computing (IJWMC)*, 1(1), Jan 2006.
26. Z. Tan, C. Lin, H. Lin, and B. Li. Optimization and Benchmark of Cryptographic Algorithms on Network Processors. *IEEE Micro*, 24(5):55–69, Sept/Oct 2004.
27. The Department of Computer Science, University of Aarhus, Denmark. CPN Tools: Computer Tool for Coloured Petri Nets. [Online]. Available: `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`, 2004.