Annales UMCS Informatica AI XII, 4 (2012) 51–67
DOI: 10.2478/v10065-012-0028-9

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Automatic detection of DoS vulnerabilities of cryptographic protocols

Urszula Krawczyk[1*], Piotr Sapiecha[1,2]

[1] *Krypton-Polska*
Al. Jerozolimskie 131 Warsaw, Poland
[2] *Department of Electronics and Information Technology, Warsaw University of Technology*
Warsaw, Poland

Abstract – In this article the subject of *DoS* vulnerabilities of cryptographic key establishment and authentication protocols is discussed. The system for computer-aided *DoS* protocol resistance analysis, which employs the *Petri* nets formalism and *Spin* model-checker, is presented.

1 Introduction

Denial of service attacks (*DoS*) limits the server abilities to respond to clients' requests. In this article, the attacks that exploit vulnerabilities of cryptographic key exchange and authentication protocols, will be considered. Such attacks took place in the past years, including those based on the *DoS* susceptibility of the *SSL*/*TLS* protocol, launched on: banks [**1**], well-known commercial services like Yahoo, Amazon and governmental sites [**2, 3**]. So this is a crucial and up-to-date problem.

The article will focus on the computational *DoS* (*CDoS*) attacks, that exhaust servers computational resources and connections queue. As the server must process all the incoming messages, it is by definition vulnerable to *DoS* attacks and can not be entirely protected from them. Yet a protocol design should prevent from the situation, when the client can easily make the server engage in expensive calculations.

Problem definition The main goal of the presented work, was designing a methodology to analyse vulnerabilities of cryptographic protocols to *DoS* attacks. It should

*U.Krawczyk@krypton-polska.com

indicate the most effective attack scenarios and allow comparing *DoS* resistance of different key establishment and authentication protocols.

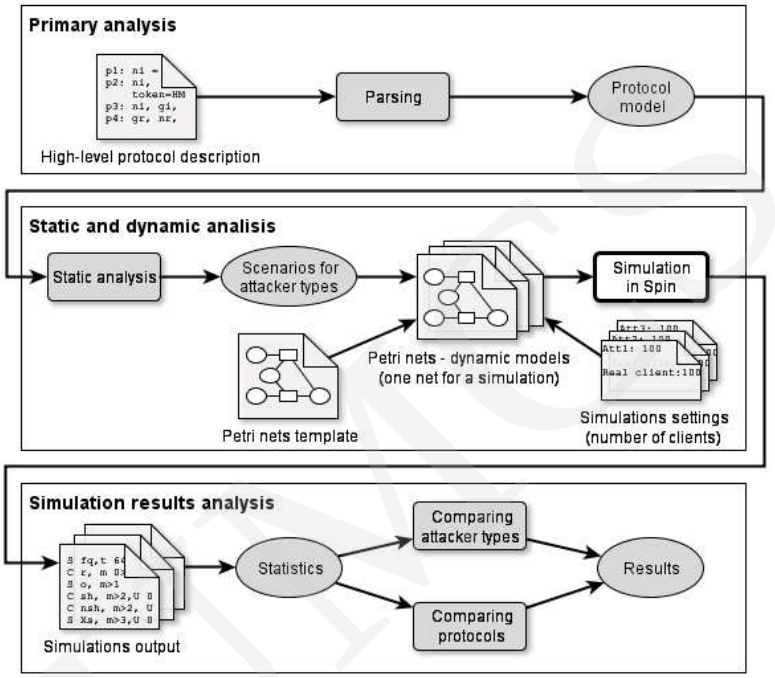


Fig. 1. Steps of analysing protocols *DoS* vulnerabilities in the *DoS Analyzer* system.

Related work Unlike the case of verifying confidentiality and integrity provided by protocols, there has been far less research in the field of designing and testing *DoS* resistance of cryptographic protocols. The most known is the Meadows's framework [**4**]. It has been used to analyse the *JFK* protocol in [**5**]. For each message, computational costs of client and server are compared. However, this approach is limited only to the *static analysis* of one protocol run and requires *manual*, work consuming calculations.

Dynamic analysis, involving many parallel protocol runs modelled as a *Petri* net, was presented in [**6, 7**]. In [**6**] first the authors present costs of protocol runs for chosen types of clients (static analysis). Transport versions of the *SSL* protocol and the *HIP* protocol are considered. Then a *Petri* net describing each protocol step is shown. It contains detailed information of every users action, which is unfortunately *redundant* during simulation. The model takes into account the *puzzle* mechanism. Yet it does not consider sharing costs by the attackers, replays of messages and reusable exponentials of the server.

The [**7**] article concentrates on slightly different aspects of protocol design, considering the broadcast *DREAM* protocol in the ad-hoc wireless network. Simulations are done in the *CPN Tools* and *Matlab*.

These methodologies are also *not automated*, as the nets are manually created for every analysed protocol.

2 Our approach

The protocol *DoS* resistance analysis methodology presented in this article is illustrated in Fig. 1. As can be seen, the *input* to the analysis process is a high-level protocol description. The *output* includes the information on the most dangerous attack types and vulnerabilities as well as comparison and evaluation of different protocols *DoS* resistance. The steps of our approach are the following:

(1) **Parsing protocol description** – parsing high-level protocol description.

(2) **Static analysis** – determining each user's step, computational and memory costs and scenarios of attacks.

(3) **Dynamic analysis** – creating an instance of a *Petri* net for each simulation. Running a simulation for each type of attack, with a chosen number of honest clients and attackers of the type under consideration.

(4) **Comparing different attack scenarios** – finding the most dangerous attack type at (which exposes vulnerabilities of the protocol), such that:

$$f(at) = \min\Big(\big\{\ f(at_j) : \quad j \in \{1 : AT\}\ \big\}\Big). \tag{1}$$

Where:

- $f(at_j)$ – the number of successful honest clients protocol runs, for a simulation with the j-th attacker type,
- AT – the number of attacker types in the modelled protocol, defined by equation (3) (in this article bounded by 15).

(5) **Comparing protocols** – for a set of P analysed protocols, finding the most *DoS*-resistant protocol p, such that:

$$f(at^p) = \max\Big(\big\{\ f(at^i) : \quad i \in \{1 : P\}\ \big\}\Big). \tag{2}$$

Where:

- at^i is the most dangerous attack type in the i-th protocol, as denoted by equation (1).

3 Protocol model for the DoS analysis

The protocol is described in a high-level language. This makes it possible to analyse protocols, which haven't been implemented yet. The language is of our own design. Its grammar is not presented here because of the limited space in the article. Yet it

can be found in [**8**]. The examples of protocol specification are shown in sections 3.2 and 5.

3.1 Static protocol model

The static protocol model includes all the details from the parsed protocol specification. A protocol run is one of the basic terms in the model. It includes protocol participants actions and states that store the actual knowledge of the users. As shown in Fig 2, in one step of a protocol run, the message sender creates all the necessary fields and sends the message. Then if this is the last message sent in the protocol, the user accepts the protocol run and establishes a session.

The receiver computes the elements needed to process the message and verifies or decrypts the message fields, according to the protocol specification. If this is the last message in the protocol and the verification is positive, then the user accepts the protocol run and establishes a session.

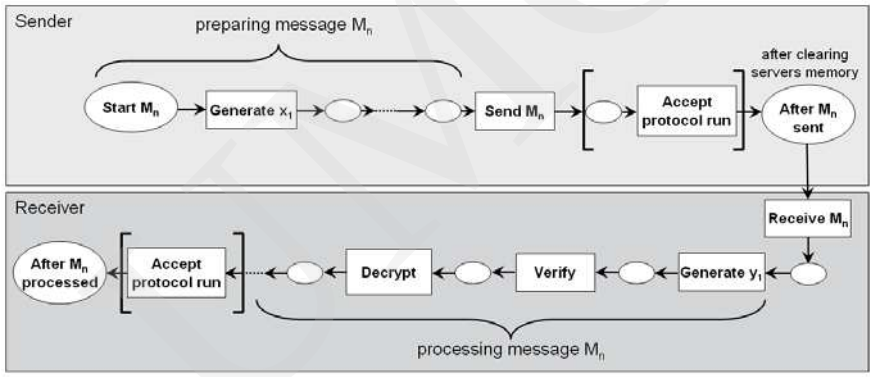


Fig. 2. Steps of senders and receivers in a protocol run.

3.2 Clients launching DoS attacks

The client that is not honest can launch a *DoS* attack in several ways. The set of attacker types is determined during the static analysis. The main properties of the attacker are:

- Aiming at exhausting server resources,
- The ability to deviate from the protocol specification by:
  - **Not verifying the received messages** – thus avoid unnecessary costs,
  - **Producing fake messages** – sending random data at no cost instead of expensive fields,
  - **Sharing computational costs** – sharing the costs among a group of attackers, which makes expensive operations cheap,
  - **Finishing protocol run** – an attacker may not respond to the server message and stop protocol run in any stage,

– **Message replay** – once generated, a message may be replayed many times at no cost,
– **Eavesdropping messages** – an attacker may capture and resend messages originally sent by legal clients.

Determining attacker types When considering the above features, there is a finite set of adversary types for a protocol. Let V be *the number of fields verified by the server* and let M be *the number of messages received by the server*. Then the size of the set of attacker types is calculated with equation (3).

$$AT = 2 \cdot V + 3 \cdot M. \tag{3}$$

The equation is derived from the fact that there are the following kinds of adversary types:

- A group of $V + M$ attacker types – each subsequent attacker type proceeds one step further in a protocol run,
- A group of $V + M$ attacker types – analogous to the previous group, except for replaying the last message (up to the server reset),
- A group of M attacker types – consists of attackers that eavesdrop a message from a real, honest client and replay it repeatedly (up to the server reset).

Examplary attacker types Fig. 3 depicts part of *STS* protocol specification, with the server verifying one field (`sig3` signature). Hence $M = 2$, $V = 1$ and there are altogether $2 * (1 + 2) + 2 = 8$ adversary types that are listed in Table 1).

```
[Packets]
p1: IDi = CERT(privI), gi = PM(g, i) // client sending his cert. and D-H exponential
// server sending his certificate, D-H exponential and signature over the exponentials
p2: IDr = CERT(privR), gr = PM(g, r), sig2 = S(privR, d2 = C(gi, gr) )
p3: sig3 = S(privI, d3 = C(gi, gr) )

[ToProcess]          // fields to be processed by the message receiver
MSG p1: ;
MSG p2: sig2 ;       // signature from MSG 2 verified by a client
MSG p3: sig3 ;       // signature from MSG 3 verified by the server
```

Fig. 3. Part of *STS* protocol specification for the *DoS Analyzer* system.

Whereas in the case of *SigmaI* protocol, the server receives three messages and verifies the following three fields: `token`, `mac3`, `sig3` (see Fig. 7; Page 62). So according to equation (3), there are $2 * (3 + 3) + 3 = 15$ adversary types (see Fig. 8; Page 63).

3.3 Dynamic protocol model

The dynamic model allows to simulate multiple, simultaneous protocol runs, while monitoring the server state. Unlike in [**6**], the model presented here includes only

Table 1. Attackers types for the *STS* protocol.

A group of $V + M = 3$ attacker types:	
Attacker type 1	Sends *MSG1* with random content and ignores *MSG2*.
Attacker type 2	Sends *MSG1* and then *MSG3* with random content – verification of `sig3` will fail.
Attacker type 3	Sends *MSG1* and *MSG3* that are successfully verified by server.
A group of $V + M = 3$ attacker types:	
Attacker type 4	Analogous to attacker 1 but keeps replaying once generated *MSG1*.
Attacker type 5	Analogous to attacker 2 but keeps replaying once generated *MSG3*.
Attacker type 6	Analogous to attacker 3 but keeps replaying once generated *MSG3*.
A group of $M = 2$ attacker types:	
Attacker type 7	Eavesdrops *MSG1* from real, honest client and repeatedly resends it.
Attacker type 8	Eavesdrops *MSG3* from real, honest client and repeatedly resends it.

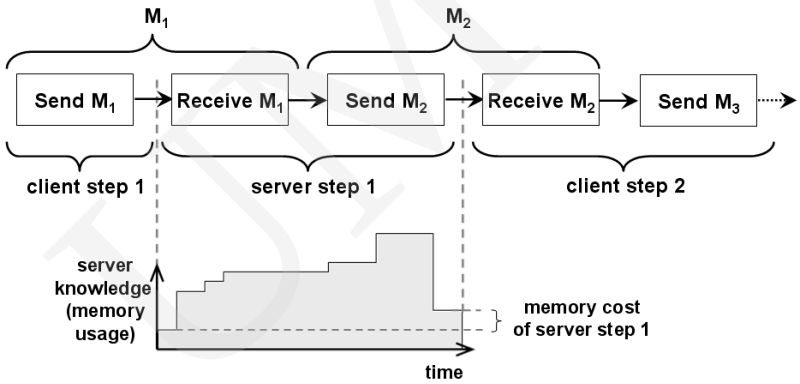


Fig. 4. Computational and memory costs of one protocol step.

information necessary for the simulation. This makes the model less complicated and speeds up simulation.

Computational costs of protocol run step Detailed information about every action of the protocol user, that is stored in the static model (Fig. 2), is redundant in the dynamic model. What is important, is the aggregated actions costs of each user step, which is illustrated in Fig. 4. The following action types costs are separately handled and stored:

- **Server costs**
  - `REUSED` – the actions of generating fields that are reused by the server (e.g. exponentials), this kind of actions is performed only periodically, after the old values are reset (which is here called the server reset),
  - `NORMAL` – the actions concerning the elements that are not reused,

– `REPLAY` – the actions performed by the server after receiving the replayed message, where:

$$REUSED \cup NORMAL = ALL\_SERVER\_COST, \quad (4)$$

$$REUSED \cap NORMAL = \emptyset, \quad (5)$$

$$REPLAY \subseteq NORMAL. \quad (6)$$

- **Client costs**
  – `SHARED` - the actions whose costs can be shared among attackers (not applicable for honest clients),
  – `NON_SHARED` - the actions whose costs can not be shared among attackers, where:

$$SHARED \cup NON\_SHARED = ALL\_CLIENT\_COST, \quad (7)$$

$$SHARED \cap NON\_SHARED = \emptyset. \quad (8)$$

Memory costs of protocol run step At the end of every user step, the elements that will not be used in subsequent steps or can be cheaply reconstructed later, are erased from memory. As can be seen in Fig. 4, the server memory cost is calculated as a difference between memory usage after the server step and before it.

3.4 Modelling protocol as a Petri net

For the purpose of simulating multiple protocol runs, high-level, coloured, timed *Petri* nets were used [**9, 10, 11**]. Fig. 5 a) depicts a graphical representation of the examplary net. Places drawn as ellipses can store tokens. Transitions are drawn as rectangles. Tokens can travel from place to place according to arc expressions, after the transition fires. The time it takes for the transition to fire is determined by the time inscription (here `@+(hd ccs)`). In the examplary net, this models the time needed by the client to receive *MSG2* and send *MSG3*.

Dynamic model structure The *Petri* nets used in the presented system have the following features:

- ***Petri* net structure** – the net consists of subpages describing behaviour of client and server, while the main page contains the instances of subpages and links them together,
- **Coloured tokens** – each token represents a protocol run with one client and bears information about:
  – client type (real client or type of attacker),
  – the number of the currently processed message (incremented during the protocol run),
  – computational and memory costs of every user's step, in the protocol run with the considered client type (as described in section 3.3),
  – delays after a client is ignored by the server or before a client starts a new protocol run.

- **Client and server subpages** – for each message in the protocol there is an instance of client and server subpage (e.g. a server subpage for receiving *MSG1* and sending *MSG2*),
- ***Petri* net template** – for each analysed protocol, a *Petri* net is created by simply parameterizing a template net with: the number of messages, the number and type of tokens,
- **Tokens movement** – tokens travel between the client and server subpages. After a particular protocol run is finished, a token moves back to the initial place before it gets engaged in a new run.

Client model Client subpage has quite a simple construction, as shown in Fig. 5 a. A token in the `MsgInNum` place determines that this subnet represents receiving *MSG2* and possibly sending *MSG3*. The client has two alternatives: responding with *MSG3* (`ComputeOutMsg` transition) or starting a new protocol run (`Finish` transition), which is available only for an attacker. The choice, cost and duration of action are controlled by the information found in the token un the `MsgIn` place.

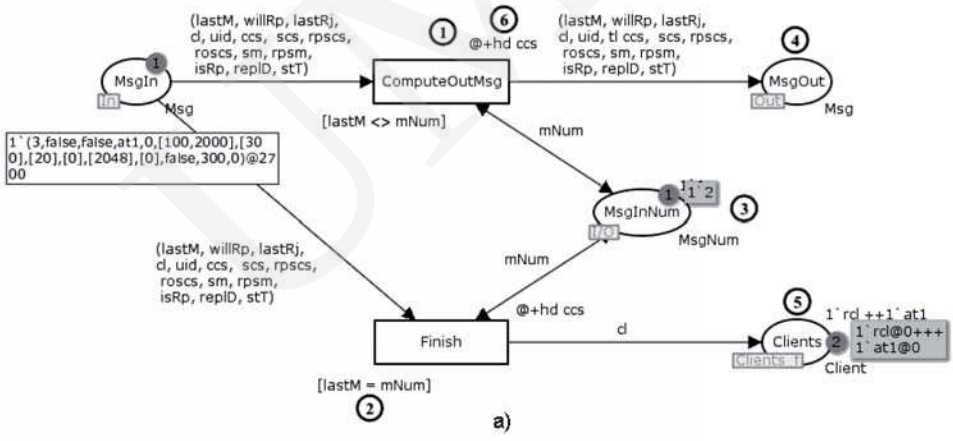


Fig. 5. Client model: a) *Petri* net graphical representation in the *CPN Tools*

Server model The *Petri* net representing the server behaviour is much bigger and more complex than the client model. Therefore it is not presented here. Instead the description of server logic is presented in the decision diagram in Fig. 6.

3.5 Protocol model properties

The protocol model, used in the presented methodology, covers the following crucial protocol features:

Fig. 5. Client model: b) textual *Petri* net specification in the *Promela* code. The numbers in the white circles indicate the corresponding elements in both models.

- **Reuse of message elements by the server** – the anti-*DoS* technique that reduces computational costs and increases *DoS* resistance of the server,
- **Sharing costs among attackers** – increases capabilities of attackers,
- **Servers computational resources** – servers computation speed is inversely proportional to the number of clients served at the time,
- **Attackers computational resources** – the more adversaries are using the shared resources at the time, the longer it will take to finish calculations for each attacker. Yet the adversary was limited to using at most four machines concurrently.

4 Petri nets simulation tool choice

Petri nets are generally represented graphically. However, in the *DoS Analyzer* system a textual net description was chosen. The well-known simulator and model-checker *Spin* [**12, 13**] was used, together with the *Promela* language. When compared with the *CPN Tools* [**14**] (which we used in the early stage of the project), *Spin* suited more for our application. It is much faster and does not require human user's interaction with GUI for each simulation, which automates the process.

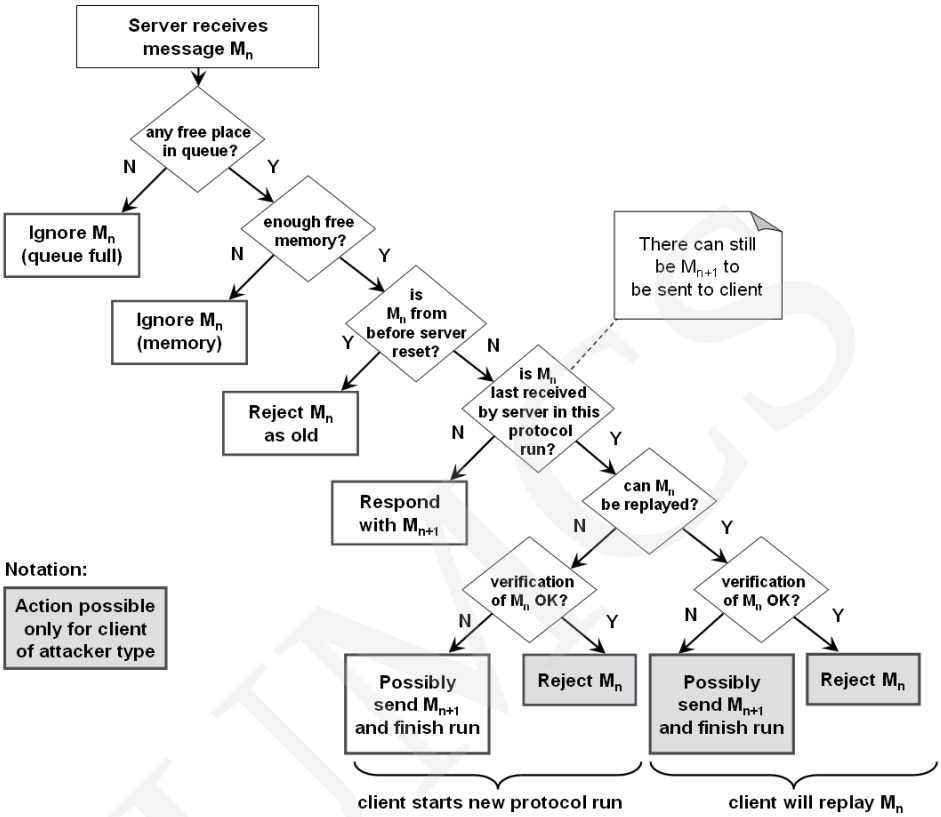


Fig. 6. Server model logic after receiving the message M_n.

The method to represent the *Petri* nets in the *Promela* language is shown in Table 2. According to these rules the net in Fig. 5 a) was transformed into the code in Fig. 5 b.

Processing simulation output During the protocol simulation, some data is written into files, including: client costs, server costs, information of server ignoring clients during *DoS* attack, honest clients establishing a session, server memory usage. This data is then aggregated by the *DoS Analyzer* system to calculate statistics.

5 Results

The protocol analysis methodology presented so far was incorporated into the *DoS Analyzer* system, that aids the process. This section will demonstrate the results of analysis of two examplary protocols: the simple *STS* protocol [**15, 16**] (described in Fig. 3) and the *SigmaI* protocol [**16, 17**] that has been equipped with the *DoS* resistance mechanisms (see Fig. 7).

Table 2. *Petri* net representation in the *Promela* language for the *Spin* simulator.

***Petri* net feature**	***Spin***
Coloured tokens	`typedef` mechanism similar to *C*-language structs
Non-determinism	Built-in nondeterministic *Promela* language constructions (`if`, `do`)
Hierarchical subnets for client and server	Asynchronic processes in *Promela* language (`proctype`)
Protocol state stored in specific places (e.g. server memory usage in `SrvMem` place)	Data kept simply in variables
Tokens travelling between client and server subnets	Structs sent via communicational channels between processes (`chan`)
Transition firing delayed by time inscriptions	When a protocol step is performed, a struct (token) is inserted into a queue managed by the additional time process. The token is passed to its destination process (message receiver or first message sender in a new protocol run), only after the necessary computations and when the specified time interval is over

Simulation configuration Both protocols were simulated for identical computational and memory resources, shown in Table 3. This makes it possible to compare the simulation results. The cryptographic algorithms used were: *AES*, *MD5* and *RSA*. The symmetric key length was 128 b, the asymmetric key length was 1024 b. Such values are generally used and recommended as currently safe by *NIST* [**18**] and *ENRYPT* [**19**]. Computational cost came from benchmarks [**6, 20**].

For each type of attacker there was a simulation with 100 honest clients and 100 attackers. Each simulation time spanned one server reset cycle (30s). Such a configuration allows to observe computational *DoS* (*CDoS*).

Table 3. Computational and memory resources configuration for the simulation.

Computational resources	
Client connection queue length at server	80 slots
Computational power of one machine used by server, client and attacker, for the purpose of protocol computations	230000 kcycle/s
The number of machines used together by adversaries	20
Maximal attacker speed up, due to distributed computations	4 times
Memory resources	
Server memory	2097152B

```
[Packets]
p1: ni = N(8), gi = PM(g, i)
p2: ni, nr = N(8), gr = PM(g, r),
    token = HMAC("SHA1", privHash,
        d2 = C(gr, gi, nr, ni, ipi = IP("remote")))
p3: ni, gi, nr, gr, token
p4: gr, nr,
    enc2 = E("DES3",
     Ke = HMAC("SHA1", ci = PM(g, i, r), nis = C(ni,nr)),
      e2 = ENC(
        IDr = CERT(privR),
        sig2 = S(privR, d1 = C(ni, gr)),
        mac2 = HMAC("SHA1", Ka = HMAC("SHA1",ci,nis), IDr)
        )
    )
p5: enc3 = E("DES3", Ke,
      e3 = ENC(
        IDi = CERT(privI),
        sig3 = S(privI, d3 = C(nr, gi)),
        mac3 = HMAC("SHA1", Ka, IDi)
        )
    )
[Cache&Reuse]  // fields that are computed only once a reset
server RESET gr EVERY 30 s;
server RESET privHash ON gr reset;
```

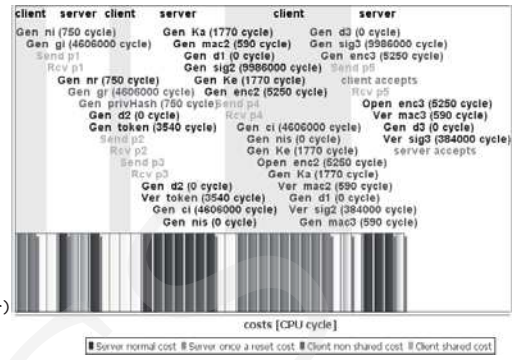


Fig. 7. *SigmaI* protocol specification fragment and protocol run visualization in the *DoS Analyzer* system.

Static analysis of SigmaI protocol Fig. 7 shows *SigmaI* protocol specification and visualization of the protocol run with an honest client. The *SigmaI* protocol includes the *cookie* mechanism [**6**]. Only a client that sent *MSG1* and received *MSG2*, can send *MSG3* with `token` field that will be accepted by the server.

The results of static analysis are depicted in Fig. 8. The first row in the table refers to an honest client, while the other rows are related to attacker types.

After the static analysis, *DoS Analyzer* application indicates (with bold face in the `Srv - cl cost` column) the most significant difference between the client and server costs in the case of adversaries of 4, 5, 10 and 11 type. These are potentially the most dangerous attack scenarios.

Dynamic analysis of the SigmaI anf STS protocols After the static analysis, which is similar to the Meadows's framework, there are some assumptions about the effectiveness of different attack types. Nevertheless only dynamic analysis can address the problem of multiple, interacting protocol runs. The results of *SigmaI* protocol model simulation are shown in Fig. 9. Each row corresponds to a simulation with the attacker type indicated in the first column. The main criteria is the number of sessions established by real clients (see section 1), found in the second column. The minimal number of successful runs was detected for adversaries 3, 5, 6 and 4. These most dangerous attacks are based on the following *SigmaI* protocol vulnerabilities:

Nr	Clients type	Cl. nonshared cost	Srv. every run cost	Srv - Cl cost	Client parameters
0	RCL	19597970	**14999050**	-4598920	last msg sent p5 (nr 5), NO REPLAY, last msg OK
1	AT1	**0**	4290	4290	last msg sent p2 (nr 2), NO REPLAY, last msg OK
2	AT2	**0**	7830	7830	last msg sent p3 (nr 3), NO REPLAY, Server FAILS at Ver token
3	AT3	**0**	14609210	14609210	last msg sent p4 (nr 4), NO REPLAY, last msg OK
4	AT4	**0**	**14615050**	**14615050**	last msg sent p5 (nr 5), NO REPLAY, Server FAILS at Ver mac3
5	AT5	**9380**	**14999050**	**14989670**	last msg sent p5 (nr 5), NO REPLAY, Server FAILS at Ver sig3
6	AT6	9995380	**14999050**	5003670	last msg sent p5 (nr 5), NO REPLAY, last msg OK
7	AT7	**0**	0	0	last msg sent p2 (nr 2), REPLAYS p1, last msg OK
8	AT8	**0**	7830	7830	last msg sent p3 (nr 3), REPLAYS p3, Server FAILS at Ver token
9	AT9	**0**	4290	4290	last msg sent p4 (nr 4), REPLAYS p3, last msg OK
10	AT10	**0**	**14615050**	**14615050**	last msg sent p5 (nr 5), REPLAYS p5, Server FAILS at Ver mac3
11	AT11	**9380**	**14999050**	**14989670**	last msg sent p5 (nr 5), REPLAYS p5, Server FAILS at Ver sig3
12	AT12	9995380	14609210	4613830	last msg sent p5 (nr 5), REPLAYS p5, last msg OK
13	AT13	**0**	0	0	last msg sent p2 (nr 2), REPLAYS p1, last msg OK
14	AT14	**0**	0	0	last msg sent p4 (nr 4), REPLAYS p3, last msg OK
15	AT15	**0**	0	0	last msg sent p5 (nr 5), REPLAYS p5, last msg OK

Fig. 8. Type of *DoS* attacks in the *SigmaI* protocol – the results of static analysis in the *DoS Analyzer* system.

Simulation file	RCL accpted	RCL que ignored	RCL mem ignored	RCL ign/accept	Srv overal cost	Srv avg mem usage	Srv avg que	Attackers overal cost	Att. avg que ...
sigmaI_3.txt__RCL-100_AT3-	**227**	**67538**	0	**297.5**	**8508338060**	396278	62.0	**5364619990**	**0.0**
sigmaI_3.txt__RCL-100_AT5-	**232**	62179	0	268.0	8455749370	185766	**63.0**	5412302190	1.0
sigmaI_3.txt__RCL-100_AT6-	**233**	59088	0	253.5	8366096300	186500	**63.0**	6936636030	1.1
sigmaI_3.txt__RCL-100_AT4-	**244**	56923	0	233.25	8471953090	187466	**63.0**	5369303870	**0.0**
sigmaI_3.txt__RCL-100_AT11	281	34657	0	123.25	8260762120	152922	**63.0**	6122659450	1.0
sigmaI_3.txt__RCL-100_AT10	319	27617	0	86.5	8430795880	167637	58.0	6901381310	**0.0**
sigmaI_3.txt__RCL-100_AT12	324	20081	0	62.0	8414236750	158036	58.0	7746766500	1.0
sigmaI_3.txt__RCL-100_AT9-	340	11414	0	33.5	8435638710	296974	62.0	7242666460	**0.0**
sigmaI_3.txt__RCL-100_AT1-	400	4938	**3769**	21.75	7473273030	**997077**	52.0	8465706000	**0.0**
sigmaI_3.txt__RCL-100_AT7-	410	6032	0	14.75	7703838320	214288	52.0	8664104770	**0.0**
sigmaI_3.txt__RCL-100_AT14	411	5785	0	14.0	7703421350	128291	52.0	8677925020	**0.0**
sigmaI_3.txt__RCL-100_AT8-	411	6678	0	16.25	7697345810	140965	55.0	8639900050	**0.0**
sigmaI_3.txt__RCL-100_AT13	413	5888	0	14.25	7689594690	128769	53.0	8649113550	**0.0**
sigmaI_3.txt__RCL-100_AT15	413	5836	0	14.25	7674983190	126428	51.0	8649113550	**0.0**
sigmaI_3.txt__RCL-100_AT2-	419	6164	0	14.75	7705400100	139205	50.0	8714856150	**0.0**

Fig. 9. Results of dynamic analysis of the *SigmaI* protocol in the *DoS Analyzer* system. The four most dangerous attack types are framed.

- **Attacker type 3** – after verifying the `token` field in *MSG3*, the server gets engaged in expensive computations (*Diffie-Hellman* and generating signature), whilst an attacker can flood server with messages created at no cost. The protocol requires from the attacker only the ability to receive a reply to *MSG1*.
- **Attacker type 4** – similar to that of adversary 3 but additionally sends *MSG5* with the random content. The server will reject a message at `mac3` verification. Employing the *HMAC* function protects the server from far more computationally expensive signature verification. For this reason, this attack is less effective than the previous one.
- **Attacker types 5 and 6** – require much more computations from the attackers but allow to engage the server in all protocol steps.

Simulation file	RCL accpted	RCL que ignored	RCL mem ignored	RCL ign/accept	Srv overal cost	Srv avg mem usage	Srv avg que	Attackers overal cost	Att. avg que...
STS_3.txt__RCL-100_AT2	**186**	85402	**0**	**459.25**	**8014092000**	240058	65.0	5007176000	**0.0**
STS_3.txt__RCL-100_AT1	**191**	87367	**0**	457.5	7957250000	**465976**	**66.0**	**4971460000**	**0.0**
STS_3.txt__RCL-100_AT3	210	**93430**	**0**	445.0	7921260000	221139	**66.0**	6143318000	1.0
STS_3.txt__RCL-100_AT6	273	58250	**0**	213.25	7921638000	168216	61.0	6782764000	1.0
STS_3.txt__RCL-100_AT5	282	52109	**0**	184.75	7879500000	160576	64.0	6102558000	**0.0**
STS_3.txt__RCL-100_AT4	313	42542	**0**	136.0	7880206000	278286	64.0	6670436000	**0.0**
STS_3.txt__RCL-100_AT8	338	41276	**0**	122.0	7771484000	145526	58.0	7249842000	**0.0**
STS_3.txt__RCL-100_AT7	340	41495	**0**	122.0	7708504000	145699	58.0	7217574000	**0.0**

Fig. 10. Results of dynamic analysis of the *STS* protocol in the *DoS Analyzer* system. The two most dangerous attack types are framed.

Specifications of the *STS* protocol and the attacker types derived from the static analysis, were earlier presented on page 56. In the case of this protocol, the following attack scenarios turned out to be the most effective:

- **Attacker type 1** – sends MSG1 with random content at no cost, whereas receiving *MSG1* causes the server to compute an expensive exponential and signature.
- **Attacker type 2** – similar to adversary 1 but additionally sends *MSG3* with random content. Receiving *MSG3* causes the server to expend computational resources for signature verification, while the attacker generates both messages at no cost. This allows to engage the server in all protocol steps. This attack type turns out to be the most effective.

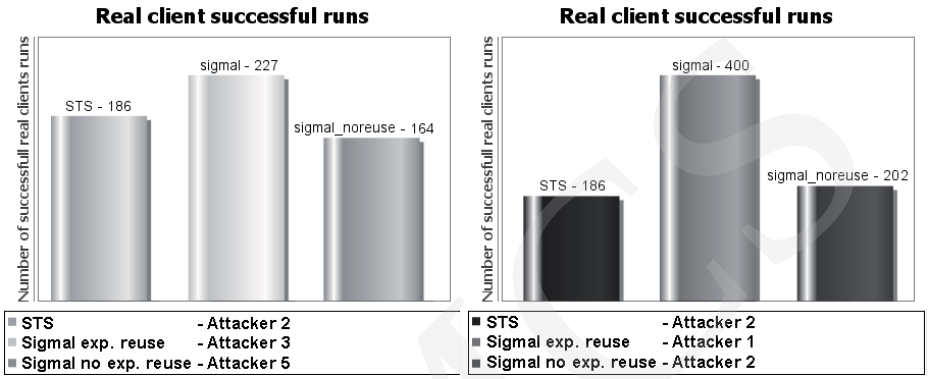
DoS vulnerabilities of the STS and SigmaI protocols The obvious shortcoming of the *STS* protocol design, is the lack of assurance that the client will expand computational resources as much as the server. Thus the protocol is significantly susceptible to *DoS* attacks.

On the other hand, the *SigmaI* protocol incorporates built-in mechanisms protecting the server from *DoS* attacks. This includes: reusing exponentials, performing inexpensive verification operations (*HMAC*) before checking the signature, employing the *cookie* scheme. Still, the *cookie* mechanism may be not strong enough to mitigate the attacks, if the adversary can receive server response in spite of *IP-spoofing*. The reason is that, sending the valid *cookie* ensures only that the client, distinguished by an *IP* address, took part in the earlier message exchange. It does not guarantee the client got engaged into computations. A solution to this might be using the *puzzle* mechanism [**6, 21**].

Comparative analysis of the STS and SigmaI protocols Each protocol is represented by the results of simulation with the most dangerous adversary type for the protocol (see Section 1). The compared protocols are: *STS* and *SigmaI* with and without exponentials reuse.

Fig. 11 a) shows a diagram with the number of sessions established by legal users for each protocol. The best performance was demonstrated by the *SigmaI* protocol with

the exponential reuse. However, the other versions of the *SigmaI* protocol turned out to be less efficient than the simple *STS* protocol. The reason for this situation might be the necessity to generate a new exponential for each protocol run, combined with the increased complexity of the protocol.



a) b)

Fig. 11. Comparative analysis of the *STS* and *SigmaI* protocols.

Additionally, a case of adversary with limited abilities was considered. The situation is when the attacker can only receive a response from the server, providing his real *IP* address. Yet without *IP-spoofing*, the attack will probably be detected by the server (*IDS*). This is quite a rational assumption. In this case in the *SigmaI* protocol, attacker type 3 can not be taken into account any more, as it demands from the client to send in *MSG3* a valid `token` field, received with *MSG2*. So the permitted attack scenarios are: 1, 2, 7, 8, 13, 14, 15. According to the simulation results in Fig. 9, attack type 1 was chosen as the most dangerous in this set. Whereas in the case of protocol version without the exponential reuse attacker type 2 was chosen. In the protocol *STS* adversaries 1, 2, 4, 5, 7 and 8 are considered, with attacker type 2 being the most effective.

The results of comparative analysis for the constrained adversaries are shown in Fig. 11 b. In such circumstances both versions of the *SigmaI* protocol presented substantially more *DoS* resistance than the *STS* protocol.

Conclusions after the protocol analysis Comparing protocols performance under the *DoS* attacks has proven the mechanisms used in the *SigmaI* protocol to be useful. Choosing a more complex design has shown to be justified. In contrast to the simple *STS* protocol, in the *SigmaI* protocol negative effects of *DoS* attacks were reduced.

6 Conclusions

We developed a nowel methodology for analysing *DoS* resistance of cryptographic key establishment and authentication protocols. The process is computer-aided with

the *DoS Analyzer* system. It allows to easily discover *DoS* vulnerabilities of protocols and identify the most dangerous attack scenarios. The system makes it also possible to compare *DoS* resistance of different protocols. Our system has been applied to the *SigmaI* protocol, which has proven the effectiveness of the anti-*DoS* techniques employed in that protocol.

Unlike the Meadows's framework [**4, 5**], our approach combines both static and *dynamic* analyses. This allows to take into account the server behaviour during many parallel protocol runs. Also when compared with the *Petri* nets introduced in [**6**], our model is generated and simulated entirely *automatically*. Additionally, the model is more concise. To summarize, the main features that distinguish our approach are:

- **Modelling the anti-*DoS* mechanisms protecting the server and specific attackers abilities**,
- **Two-stage analysis** – the preliminary stage of static analysis lets minimize the final dynamic model, as the data redundant during the simulation is not included, which speeds up the process,
- **Analysis automatization** – the methodology was fully automatized and the human user has only to supply high-level protocol specification and simulation configuration.

Extending the system The presented system can be improved in the following ways:

- Taking into account that also honest clients can reuse exponentials,
- Adding an option to define the clock time unit of the simulation, which would help balance the accuracy/speed issue,
- Live visualization of clients actions and server resources during simulation.

References

[1] Headlines, Bank of America Hit By Anonymous DDoS Attack, (27.12.2010); www.infosecisland.com

[2] Adair S., Pushdo DDoS'ing or Blending In?, (2010); www.shadowserver.org/wiki/pmwiki.php/Calendar/20100129

[3] Moore D., Shannon C.,The Spread of the Code Red Worm (crv2) (2001); www.caida.org/analysis/security/codered/coderedv2_analysis.xml

[4] Meadows C., A Cost-Based Framework for Analysis of Denial of Service in Networks (2001); citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.9001&rep=rep1&type=pdf

[5] Smith J., Gonzalez-Nieto J. M., Boyd C., Modelling Denial of Service Attacks on JFK with Meadows Cost-Based Framework (2006); citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.3877

[6] Tritilanunt S., Boyd C., Foo E., Gonzalez Nieto J. M., Using Coloured Petri Nets to Simulate DoS-resistant protocols (2006); www.daimi.au.dk/CPnets/workshop06/cpn/papers/Paper15.pdf

[7] Vanek T., Rohlik M., Model of DoS Resistant Broadcast Authentication Protocol in Colored Petri Net Environment, (2010); www.ic.uff.br/iwssip2010/Proceedings/nav/papers/paper_85.pdf

[8] Sapiecha P., Krawczyk U., DoS Analizer language syntax (2012); www.krypton-polska.com/upload/1f0e3dad99908345f7439f8ffabdffc4.pdf

[9] Balbo G., Desel J., Jensen K., Reisig W., Rozenberg G., Silva M., Introductory Tutorial - Petri Nets (2000); www.informatik.uni-hamburg.de/TGI/PetriNets/introductions/pn2000_introtut

[10] Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, annual workshops in years 1998-2008; www.daimi.au.dk/CPnets
[11] Jensen K., An Introduction to the Theoretical Aspects of Coloured Petri Nets (1994); www.dsc.ufcg.edu.br/ãbrantes/CursosAnteriores/MVSRP/rex.pdf
[12] Holzmann G. J., Spin model-checker; http://spinroot.com
[13] Sapiecha P., Krawczyk U., Effective reduction of cryptographic protocols specification for model-checking with Spin, Annales UMCS, Informatica AI 11 (3) (2011): 27; DOI: 10.2478/v10065-011-0002-y
[14] Jensen K., CPN Tools, www.daimi.au.dk/CPNTools
[15] Boyd C., Mathuria A., Protocols for authentication and key establishment, Springer (2003).
[16] Krawczyk H., SIGMA: the 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and its Use in the IKE Protocols (2003); ftp://ftp.pwg.org/pub/pwg/wbmm/security/sigma.pdf
[17] Bitan S., Krawczyk H., SIGMA: the 'SIGn-and-MAc' Crypto rationale and proposals – for IETF meeting (2001); www.ietf.org/proceedings/52/slides/ipsec-9.pdf
[18] Barker E., et. all, Computer security - Recomendation for key management (2007); csrc.nist.gov
[19] Giry D., BlueKrypt - Cryptographic Key Lenght Recomentation (2010); www.keylength.com/en
[20] Dai W., Crypto++ 5.2.1 Benchmarks (16.01.2011); www.cryptopp.com
[21] Beal J., Shepard T., Deamplification of DoS attacks via puzzles (2004); web.mit.edu/jakebeal/www/Unpublished/puzzle.pdf