CHAPTER 4

SYSTEM ARCHITECTURE AND DESIGN

4.1. Overview

This chapter explains about SAAIDS architecture and design. Firstly, three proposed solutions are discussed which began with description of architecture of SAAIDS including system components and roles. There are six components which are designed to do specific tasks followed by intrusion detection, agent security, agent communication protocol and agent verification protocol including their protocol design, description and algorithm.

4.2. The Architecture of SAAIDS

SAAIDS architecture has been designed to address all the problems mentioned before along with providing secure agent communication and protection of the agent itself. In achieving these goals SAAIDS architecture is developed based on P2P connection structure and this structure produced an idea of enrolling all components in an agent. There are six main components in SAAIDS architecture as shown in Figure 4.1. This architecture represents the overview of an agent in doing intrusion detection.

4.2.1. Security

To ensure secured communication between agents and protecting agent itself, this system consists of three subcomponents which are being discussed here:

a. Authentication

Authentication is a process of verifying a claimed identity. Authentication is used to avoid unauthorized personal from breaking into system. There are several issues in authentication such as forgotten password, password guessing, password spoofing and compromise of the password file. All these issues can be solved with a good password management and protection of the password file. Technologies used are Single Sign-On and Java Authentication and Authorization Service (JAAS).

b. Encryption

Encryption is a process of protecting the confidentiality of data sent in the network. Asymmetric encryption which is also known as Public Key (PK) cryptosystem transforms plaintext into ciphertext using a secret key and an encryption algorithm (Stallings, 2007). All agent communication will be protected by a widely-known encryption technology; Elgamal Encryption.

c. Digital Signature

Digital signature is defined as data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery. Digital signature is used in installing a new agent and agent verification process. Agent verification process uses Elgamal Digital Signature integrated with SHA1 called "SHA1withElgamal".

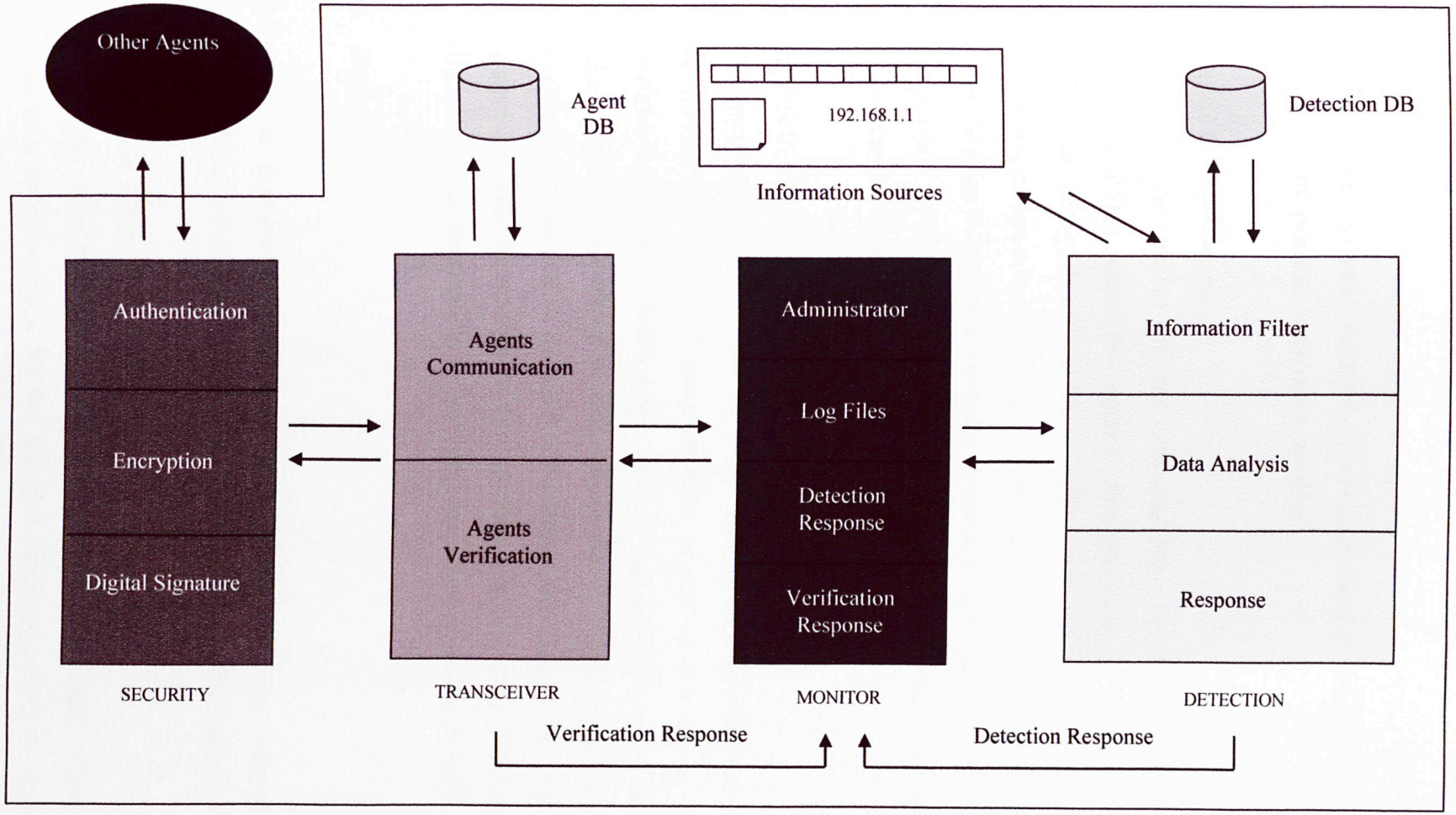

Figure 4.1: SAAIDS Architecture

4.2.2. Transceiver

Transceiver is a communication interface between agents after going through Security component. There are two subcomponents involved:

a. Agents Communication

Agents Communication initiates communication between agents which two roles; controlling agents and agents' data processing. Controlling agents is defined as doing starts and stops agents as per required. Agent's data processing means doing data controls each time received and determine where to the data has to be sent and or to be stored in agents' database. All these tasks are defined as Agents Communication Protocol and secured by encryption technology in Security components.

b. Agents Verification

To ensure that all agents running in the system are authorized agent, agents verification involves verification on each agents on time-based. When an unauthorized agent is detected in a system, Agent Verification Protocol will take action such as warning all agents about the intrusion by sending an alert to Verification Response in Monitor to broadcast the event to all agents in the system and kills connection to the intrusion source. Agent verification uses digital signature mechanism in Security components in doing its task.

4.2.3. Detection

SAAIDS is a distributed agent-based IDS which runs autonomously and independently. Detection is done its task through three phases in fundamental of IDS:

a. Information Filter

Information filter's role is to gather only information needed to be analyzed from information sources. All the gathered information will be sent to Data Analysis to be analyzed and stored in Detection DB.

b. Data Analysis

Data analysis role is to process information received from Information Filter and makes decision whatever attack threats has occurred or not. Data analysis technologies which have been determined to be used are Signature Analysis as mentioned in section 2.6.

c. Response

If attacks or threats are detected in Data Analysis, it will be sent to response level. Response will take actions on this event such as sending an alert to all agents through Detection Response in Monitor. Besides, all information gathered will be stored in Detection database as log files for future analysis.

4.2.4. Monitor

There are four modules which can be monitored through a graphical user interface (GUI) in this system. The modules are:

a. Administrator

Administrator module or also called Agent Monitoring System which shows overview of systems including agent's location and status. Besides that, this module provides controlling agent's abilities and monitoring for administrator.

b. Log Files

There are three log files in this system including:

i. Detection Log

Information filtering, data analysis and detection response information are shown in this log for reports and future analysis by administrator.

ii. Communication Log

Agent communication information is shown in this log for reports and future analysis by administrator.

iii. Verification Log

Agent verification and response information are shown in this log for reports and future analysis by administrator.

c. Detection Response

This module triggered each time attack threats are detected by Detection to all agents for further actions.

d. Verification Response

This module triggered each time fake agent is detected by Agent Verification to all agents for further actions.

4.2.5. Database

There are two databases in an agent which include:

a. Agent Database

Agent database contains all information about an agent which represents as input for Agent Verification Protocol code (VID).

b. Detection Databases

Detection database contains all information about detection done along with log and history of the detection.

4.2.6. Information Sources

Three sources in information gathering are used in this system which included IP address, operating system log and network packet.

4.3. Intrusion Detection

In SAAIDS, intrusion detection is done by a component called Detection which includes Information Filter, Data Analysis and Response. There are various sources and ways to do intrusion detection and many researches done to achieve the same objective. As mentioned in Chapter 2 before, Snort is already equipped to do intrusion detection in SAAIDS. Besides that, it is used to log for intrusion detection too. There are two facilities used in this system which are alert and log.

4.3.1. Alert Module

There are general mode of output when an alert is generated as mentioned in section 2.5.3.2,. Actually, there are four alert types option provided by Snort which is alert_fast, alert_full, alert_syslog and alert_CSV. In SAAIDS intrusion detection process used alert_fast which contains straightforward and needed information in providing intrusion response. On the other hand, alert_fast does not consume much memory in converting packet headers to ASCII or writing them to the output file. The syntax used in snort.conf file to specify alert_fast output module is:

```
output alert_fast: snort.log
```

Fast alerts show the flavor of attack, its classification, source, destination and timestamp. The sample output for alert_fast:

```
02/26- 17:59:01635549 [**] [1:2003:2] MS-SQL Worm propagation attempt [**] [Classification: Misc Attack} [Priority: 2] [UDP] Y.Y.250.124:1162 -> X.X.2.27:1434
```

To ensure that all attacks classification in SAAIDS is successfully detected, Snort's standard classifications are used which includes its classtype, description and priority. The standard classifications included in Snort are listed in Table 4.1 below which leveled with 3 default priorities; priority 1 is most severe priority level of the default rule set and priority 4 is least severe (Caswell, 2004).

Table 4.1: Snort Default Classification

Classtype	Description	Priority
attempted-admin	Attempted Administrator Privilege Gain	high
attempted-user	Attempted User Privilege Gain	high
user shellcode-detect	Executable code was detected	high
successful-admin	Successful Administrator Privilege Gain	high
successful-user	Successful User Privilege Gain	high
trojan-activity	A Network Trojan was detected	high
unsuccessful-user	Unsuccessful User Privilege Gain	high
web-application-attack	Web Application Attack	high
attempted-dos medium	Attempted Denial of Service	medium
attempted-recon	Attempted Information Leak	medium
bad-unknown	Potentially Bad Traffic	medium
denial-of-service	Detection of a Denial of Service Attack	medium
misc-attack	Misc Attack	medium
non-standard-protocol	Detection of a non-standard protocol or event	medium
rpc-portmap-decode	Decode of an RPC Query	medium
successful-dos	Denial of Service	medium
successful-recon-largescale	Large Scale Information Leak	medium
successful-recon-limited	Information Leak	medium
suspicious-filename-	detect A suspicious filename was detected	medium
suspicious-login	An attempted login using a suspicious username was detected	medium
system-call-detect	A system call was detected	medium
unusual-client-port-connection	A client was using an unusual port	medium
web-application-activity	Access to a potentially vulnerable web application	medium
icmp-event	Generic ICMP event	low

misc-activity	Misc activity	low
network-scan	Detection of a Network Scan	low
not-suspicious	Not Suspicious Traffic	low
protocol-command-decode	Generic Protocol Command Decode	low
string-detect	A suspicious string was detected	low
unknown	Unknown Traffic	low

Using standard Snort default classification, SAAIDS enable to get filtered information and alert for various attacks as mentioned in SAAIDS attacks classification. When an alert occurred, all the information received by system will be logged into Detection database, DetectionDB immediately for intrusion response process which is then sent to Detection Response in Monitor by Response in Detection for further action. Then, Detection Response will broadcast alert warning to all agents using Agent Communication which will be further discussed later in section 3.5.

4.3.2. Log Module

SAAIDS logs intrusion detection information into Detection Log which is stored in DetectionDB. Snort provides log facility which enables the information output to be stored into database. The configuration of this module is as follows:

output database: <log | alert>. *<database type>* , *<parameter list>*

The |, pipe between <log | alert>, log or alert to be used indicating that either log or alert to be used as the logging method. In this system, alert has been chosen as logging method which is being configured as a Snort rule providing two functions available; the alert facility and log facility. This means that, using alert as logging method which enable alert module functions and at the same time data is also written to the log facility.

<database type> indicates chosen database used to store the detection log. In SAAIDS, MySQL database is being used for DetectionDB. While *<parameter list>* indicates

parameter specified for the database. The following parameters are available (Caswell, 2004):

a. host - Host to connect to. If a non-zero-length string is specified, TCP/IP communication is used. Without a host name, it will connect using a local domain socket.
b. port - Port number to connect to at the server host, or socket filename extension for domain connections.
c. Dbname - Database name user Database username for authentication.
d. password - Password used if the database demands password authentication.
e. sensor_name - Specify name for this snort sensor. If sensor name is not specified, a name one will be generated automatically.
f. encoding - type of data represented in database.
g. detail – Complexion of much detailed data to be stored.

Since the packet payload and option data is binary, there is no one simple and portable way to store it in a database. There are several options of encoding the output which are HEX, BASE64 and ASCII. This system used standard ASCII as encoding option to be logged into DetectionDB which are **r**epresents binary data as an ascii string. ASCII is very good for searching for a text string impossible if you want to search for binary and readable for human.

As mentioned before, alert_fast is used in rules as alert module. In alert_fast log module, the complexion of data stored also used fast logging which represents only timestamp, message signature, source IP address, destination IP address, source port, destination port, tcp flags and protocol. Instead of full logging which contains all information of a packet including TCP/IP options and the payload, fast logging are more understandable and consumes less size of database.

4.4. Agent Security

An agent is a persistent goal-oriented entity that may move between hosts (environment – worlds) in response to changes in requirements such as security, efficiency and cost (Hou et al., 2006 (a)). Agent-based system is a multi agents platform system which using agents as doer of its tasks. All tasks and process are depending on it agents, therefore agents must be capable of autonomous actions at host (in environment) in order to satisfy its design objectives (Wiley, 2002). Agent-based Intrusion Detection System (IDS) is an IDS using agents as its platform in doing detection and other management tasks. It is important to ensure that agents are truly secured towards successive of doing its tasks.

An agent is designed by its role or tasks. In agent-based IDS, the main task for an agent is to detect for any signs of intrusion. Besides its role in detection, agent in agent-based IDS is designed to take care of its own protection. Agent security as defined by Stojkovic and Huo, can be split into two components: agent data security and agent action security (Hou et al., 2006 (a)).

Table 4.2: Agent's Attribute

Name	Data Type	Example
ID	CHAR(10)	1
Name	CHAR(10)	007
DateTimeCreated	Timestamp	19830905132800
HostIP	CHAR(10)	10.8.33.33
HostName	CHAR(10)	loque
HostAddress	CHAR(20)	saaids.loque
GroupName	CHAR(10)	loquegroup
AdministratorID	INTEGER	1

Agent data security means avoiding data from being stolen or damaged by worms or Trojan. Agent has to be protected by special key and in this case date and time created have been chosen as a main key. This main key is stored along with other attributes in an agent. The attributes of an agent is shown in Table 4.2. As usual, agentID as a primary key, hostIP as a common attributes of an agent but for date and time is very unique which every agent has unique date and time of created or born. These two

attributes are common attributes in a database system but for this purpose it can be very valuable.

How date and time can be so valuable attributes in agent security? Date and time has several special data type and format which are vary to be predicted for an outsider of an agent development. The data type and format used will be only knows by insiders. For example, in MySQL there are 6 data type and format for DATETIME, DATE, TIME and TIMESTAMP and shown in Table 4.3.

Table 4.3: MYSQL Data Type and Format for DATETIME, DATE, TIME and TIMESTAMP

Data Type / Format	Example
String YYYY-MM-DD HH:MM:SS YY-MM-DD HH:MM:SS	98-12-31 11:30:45 98.12.31 11+30+45 98/12/31 11*30*45 98@12@31 11^30^45
String YYYY-MM-DD YY-MM-DD	98-12-31, 98.12.31 98/12/31, 98@12@31
String YYYYMMDDHHMMSS YYMMDDHHMMSS	19970523091528 970523091528
String YYYYMMDD YYMMDD	19970523 970523
Number YYYYMMDDHHMMSS YYMMDDHHMMSS	19830905132800 830905132800
Number YYYYMMDD YYMMDD	19830905 830905

Date and time of the agent will be main key which will be hashed/digested using SHA1 algorithm along with other attributes to be Verification ID (VID). As example, the following hash process using Number/YYYYMMDDHHMMSS data type and format:

H(**DATETIME**+ID+Name+Others..)

=> H(19830905132800+1+007+Others..)

=> H'(VID)

This H'(VID) will be used in agent verification protocol which will be discussed later in section 3.6.

4.5. Agent Communication Protocol and Algorithm

Agent Communication Protocol is a defined procedure of message sending process in SAAIDS. This protocol is located in Transceiver along with Agent Verification Protocol. Discussion on Agent Communication Protocol is described as follows:

4.5.1. Protocol Description

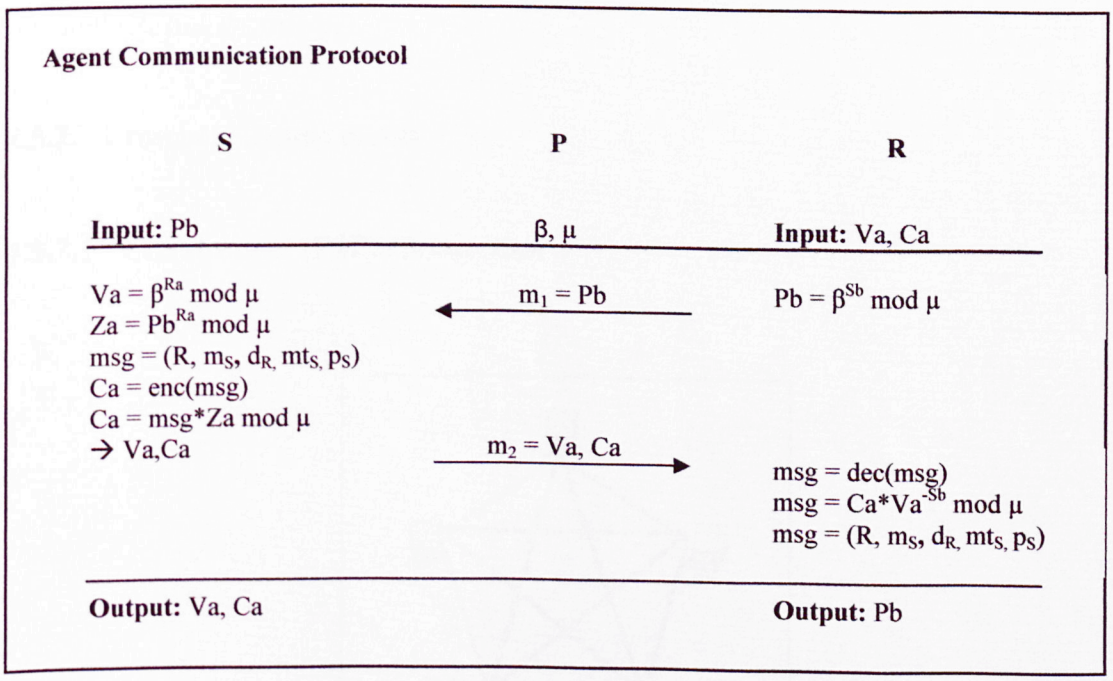

Figure 4.2: Agent Communication Protocol

In designing Agent Communication Protocol, the researcher assumes that agents connected each other by P2P connection and message queue procedure is followed. This protocol used Elgamal Encryption Algorithm to encrypt messages sent between

agents and invoked via enc(encrypt) at the sender(S) and dec(decrypt) at the receiver(R). B is a publicly known random number that serves as the generator and μ is a publicly (P) known prime number that serves as the modulus. Input for S is Pb and (Va and Ca) for R. Messages sent is defined as m_i.. Firgure 4.2 shows Agent Communication Protocol. The protocol proceeds as follows:

Step 1. R generates a static secret number S and computes static public number Pb as β^{Sb} mod μ. R receives Pb as output.

Step 2. S generates ephemeral secret key Ra and computes ephemeral public key Va as β^{Ra} mod μ. Then, S computes ephemeral encryption key Za as Pb^{Ra} mod μ. S and continues with generates msg = (R, m_S, d_R, mt_S, p_S) and encrypt, enc to Ca as msg*Za mod μ. The message sent (R, m_S, d_R, mt_S, p_S) defined as (Receiver, message of S, description of R, type of message S, priority of message S). S receives Va and Ca as output.

Step 3. R decrypt, dec the message and generates message msg = $Ca*Va^{-Sb}$ mod μ and resulted (R, m_S, d_R, mt_S, p_S).

4.5.2. Protocol Requirements

4.5.2.1 Peer-to-peer (P2P) Connection

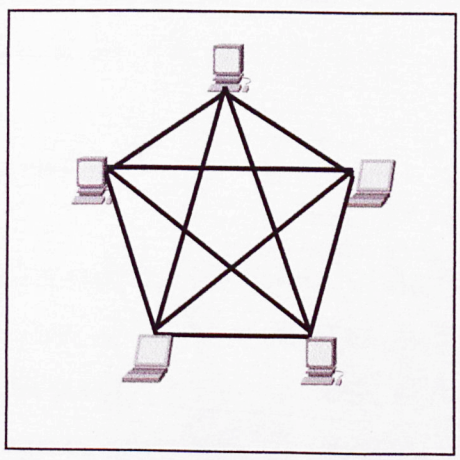

Figure 4.3: P2P Connection

In avoiding single-point of failure, SAAIDS is using peer-to-peer (P2P) connection between agents as system structure. Therefore, full depending to monitor or transceiver is avoided. Here, agents doing its own tasks, communicate each other in detecting intrusion and integrate each other when response to detection.

For multilevel authorization problem in hierarchical structure, the P2P connection as system structure simplified the complicated authorization level. Authorization processes in SAAIDS is only involving authorization on agents which occurred at agents monitoring system. Monitoring system here is defined as determining agent's location and condition which simulated in system's graphical user interface (GUI). Monitoring system will further discussed in section 3.8. Therefore, SAAIDS structure has no hierarchical structure towards avoiding multilevel authorization problem and use very limited size data storage for every agent to store authorization information. Besides that, authorization in monitoring system will not affects any running processes in the system.

In SAAIDS, there is no delay on information sending between information gathered, data analysis for intrusion and response level because all of the components play its role as an agent. In intrusion response broadcasting, no delay is expected due to system structure provides direct connection between agents. Therefore, no obstacles will cause delay on intrusion information sending and further action on intrusion response such as warning all agents about the intrusion and kill connection with the intrusion source can be done instantly. This feature ensures effectiveness guaranteed on the autonomous agents in doing its tasks in the system.

4.5.2.2 Secured Communication

Agent Communication in SAAIDS architecture is responsible to ensure that message sending between agents are secured. Encryption technology used to protect confidentiality of data in the network (Gollman, 2006). Here, it is defined as message communication between agents. Encryption is defined as a process of transforming readable data into an unintelligible form (Kamaruzzaman, 2007). All messages will be encrypted using widely known Elgamal encryption algorithm which before being sent

from one agent to others. This encryption process is very important for every communication to avoid from being damaged or data stolen. Figure 4.4 shows the overview of Elgamal Encryption. Agent communication process description using this algorithm will be further described in Agent Communication Protocol and Algorithm, activity diagram and sequence diagram in section 3.8.

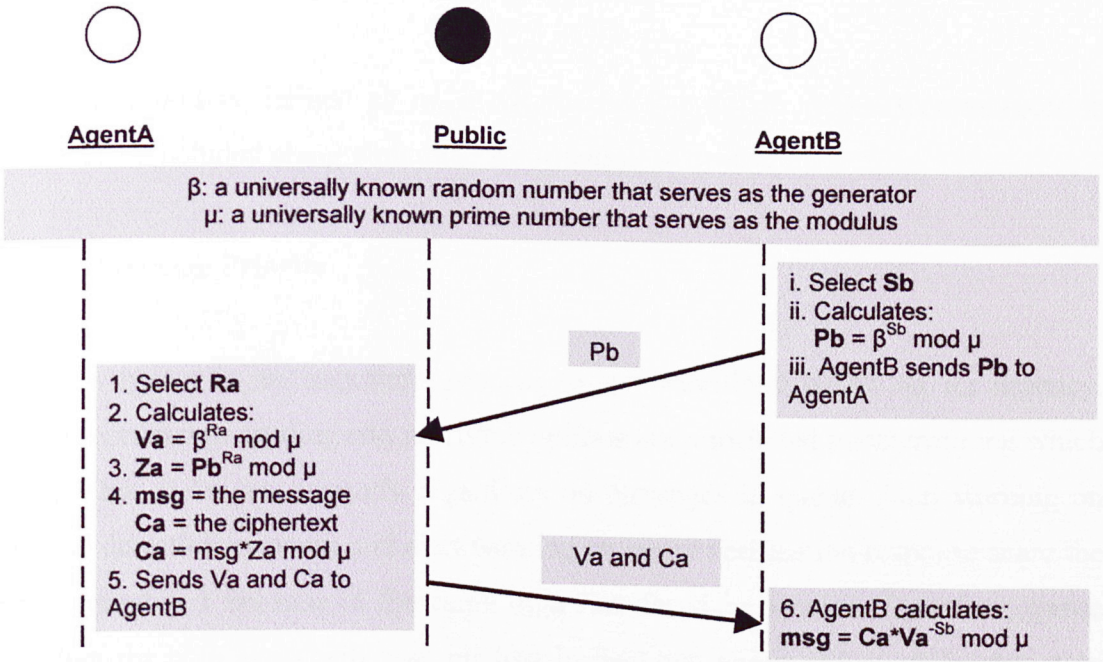

Figure 4.4: Overview of Elgamal Encryption

4.5.2.3 Message Sending and Receiving Process

Agent communication plays an important role in ensuring successive process of agent verification. Agent communication here is defined as communication processes of message sending and receiving between agents in terms of:

a. Sending and receiving detection message for intrusion detection process defined as $mdet_i$.
b. Sending and receiving detection result between agents to be logged defined as $mderst_i$.
c. Sending and receiving alert warning on response of intrusion detection to other agents defined as $mdersp_i$.

d. Sending and receiving verification message for agent verification process defined as $mver_i$.

e. Sending and receiving verification result between agents to be logged defined as $mverst_i$.

f. Sending and receiving alert warning on response of unauthorized agent detection to other agents defined as $mversp_i$.

Messages option is defined as m_i in (R, m_i, d_i, mt_i, p_i) in Agent Communication Protocol and included along with other information to be sent.

4.5.2.4 Message Priority

Messages queue is an important process to be prioritized based on its urgency. Therefore, message sending and receiving options are prioritized in determining which message has to be sent urgently regardless on messages in queue. Alert warning on intrusion detection response and alert warning on agent verification response share the priority number 1 because of the same high risk faced by system. Therefore, queue procedure for both alerts will base on first in first out system. Meanwhile, detection result and verification Messages priority is defined as p_i in (R, m_i, d_i, mt_i, p_i) in Agent Communication Protocol and included along with other information to be sent. Table 4.4 shows Message Priority which have been defined in the system.

Table 4.4: Message Priority

Priority	Variable	Message Type
1	$mdersp_i$	Alert warning on intrusion detection response
1	$mversp_i$	Alert warning on agent verification response
2	$mdet_i$	Detection message for intrusion detection process
2	$mver_i$	Verification message for agent verification process
3	$mderst_i$	Detection result to be logged
3	$mverst_i$	Verification result to be logged

4.5.3. Algorithm Description

In designing Agent Communication Algorithm, the researcher assumes that agents connected each other by P2P connection and message queue procedure is followed. Agent Communication Algorithm has two functions which are enc() and dec(). Firstly, B and μ generated for both functions as public random and prime numbers. Function enc() uses B and μ to computes Va and receives Pb as input parameter and used to compute Za if only Pb is true or else failure(Pb == false) will be returned. This condition used to ensure that Pb is truly received Then, condition for used to ensure only one process of encryption occurred each time. Only when i = 0 to i +1, msg[i] will be encrypted or else failure(i > 0) will be returned. Before that, Va and Za computed to be used to computes Ca as cyphertext which involving msg, Za and μ. The msg[i] created is including (R, m_i, d_i, mt_i, p_i) which represents (Receiver, message, description, type of message, priority of message). Finally, enc() returns Va and Ca.

In other side, function dec() receives (Va, Ca) as input to create (R, m_i, d_i, mt_i, p_i) from msg[i] which involving Ca, Va, Sb and μ. Before that, condition if used to ensure that Va and Ca are truly received or else failure(Va, Ca == false) will be returned. Then, condition for used to ensure only one process of decryption occurred each time. Only when i = 0 to i +1, msg[i] will be decrypted or else failure(i > 0) will be returned. Agent Communication Algorithm is shown in Figure 4.5.

```
Agent Communication Algorithm

Generates B, μ

enc(Pb)
{
    If (Pb == true)
    {
        for(i = 0 to i + 1)
        {
            Computes Va ← β^Ra mod μ
            Computes Za ← Pb^Ra mod μ
            Create msg[i] ← (R, m_i, d_i, mt_i, p_i)
            Ca ← msg[i]
            Ca ← msg*Za mod μ
            i++
            return Va, Ca
        }return failure(i > 0)
    }return failure(Pb == false)
}

dec(Va, Ca)
{
    If(Va, Ca == true)
    {
        for(i = 0 to i + 1)
        {
            msg[i} ← Ca*Va^-Sb mod μ
            Create (R, m_i, d_i, mt_i, p_i) ← msg[i]
            i++
            return success (msg[i] == true)
        }return failure(i > 0)
    }return failure(Va, Ca == false)
}
```

Figure 4.5: Agent Communication Algorithm

4.6. Agent Verification Protocol and Algorithm

Agent verification is defined as:

a. A process of making sure that all agents running in the system are authorized agents.

b. A process of tracing unauthorized agent running in the system.

The process involves peer to peer connection (P2P) to enable direct connection between agents and ensure fast and reliable verification message sending and no obstacles between agents during the verification process.

4.6.1. Protocol Description

In designing Agent Verification Protocol, the researcher assumes that agents connected each other by P2P connection and message queue procedure is followed. This protocol used SHA1 with Elgamal Digital Signature Algorithm to digitally sign between agents to ensure that all agents running in system are authorized agent. There are exchange protocol and response protocol in Agent Veification Protocol. In this protocol, there are S as sender and R as receiver.B is a publicly known random number that serves as the generator and μ is a publicly (P) known prime number that serves as the modulus while H() is a known hash function. A message sent is defined as m_i. Figure 4.6 shows Agent Verification Protocol. The protocol proceeds as follows:

Step 1. S in exchange protocol generates a static secret number Sb and computes static public number Pb as β^{Sb} mod μ. Then, S generates ephemeral secret key Rb and computes ephemeral public key Vb as β^{Rb} mod μ. S continues with generates msg (R, m_R, d_S , t_S) hash to hi along with Vb. (R, m_R, d_S , t_S) is defined as (Receiver, message to R, description of S, time of S). After that, S computes sign as Rb + hi * Sb mod μ - 1. S receives Pb, Vb, msg and sign as output.

Agent Verification Protocol

exchange protocol

S	P	R
Input:	β, μ,H()	**Input:** Pb, Vb, msg, sign
$t == t_S$? : continue $Pb = \beta^{Sb} \bmod \mu$ $Vb = \beta^{Rb} \bmod \mu$ $msg = (R, m_R, d_S, t_S)$ $hi = H(msg \| Vb)$ $sign = Rb + hi * Sb \bmod \mu$ - 1 → Pb, Vb, msg, sign	m_1 = Pb, Vb, msg, sign →	$hi' = H(msg \| Vb)$ $msg = (S, m_R, d_S, t_S)$ verify (sign) $\rightarrow \beta^{sign} \bmod \mu ==$ $Vb\ Pb^{hi'} \bmod \mu$? : *verified* $\rightarrow \beta^{sign} \bmod \mu =/=$ $Vb\ Pb^{hi'} \bmod \mu$? : *unverified*
Output: Pb, Vb, msg, sign		**Output:**

response protocol

Input: $mveri_R$		**Input:**
$t => t_R$? : verified $log(R, d_R, t_R)$ $mveri_S = (i, d_S, t_S)$ → $mveri_S$	$m_2 = mveri_R$ ←	← $mveri_R = (R, d_R, t_R)$
$t == null \| t =/= t_R$? : unverified $kill(R, d_R, t_R)$ $log(R, d_R, t_R)$ $mversp_S = (i, d_R, t_R)$ → $mversp_S$		
Output: $mveri_S \| mversp_S$		**Output:** $mveri_R$

Figure 4.6: Agent Verification Protocol

Step 2. R in exchange protocol generates hi' as H'(msg' || Vb) and verify sign. (S, m_R, d_S , t_S) defined as (Sender, message to R, description of S, time of S). If β^{sign} mod μ = Vb $Pb^{hi'}$ mod μ, verified procedure will proceeds. Otherwise if β^{sign} mod μ =/= Vb $Pb^{hi'}$ mod μ, unverified procedure will proceeds.

Step 3. If only sign verified, S in response protocol will receives verification message $mveri_R$ = (R, d_R, t_R) defined as (Receiver, description of R, time of R) from R and log(R, d_R , t_R) which log the verification response into verification log. Then, S will generates $mveri_S$ = (i, d_S , t_S) defined as (other agents, description of S, time of S) and sends it to other agents.

Step 4. If sign is unverified or S do not receives verification message $mveri_R$ = (R, d_R, t_R) in specific time, S will generates procedure kill(R, d_R , t_R) defined as (Receiver, description of R, time of R) which kill connection with R and log(R, d_R , t_R) which log the verification response into verification log. Finally, S generates $mversp_S$ = (i, d_R,t_R) defined as (other agents, description of R, time of R) and sends it to other agents to do kill procedure.

4.6.2. Protocol Requirements

4.6.2.1 Digital Signature

As mentioned above, Elgamal Encryption is used in agent communication as message encryption to ensure secure message sending between agents in the process of agent verification. If Elgamal encryption is used as message encryption, message sent between agents will be signed by Elgamal Digital Signature. Digital signature is defined as data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (Stallings, 2007). Besides agent verification process, digital signature is also used in installing new agent in the system. Figure 4.7 shows the overview of Elgamal Digital Signature.

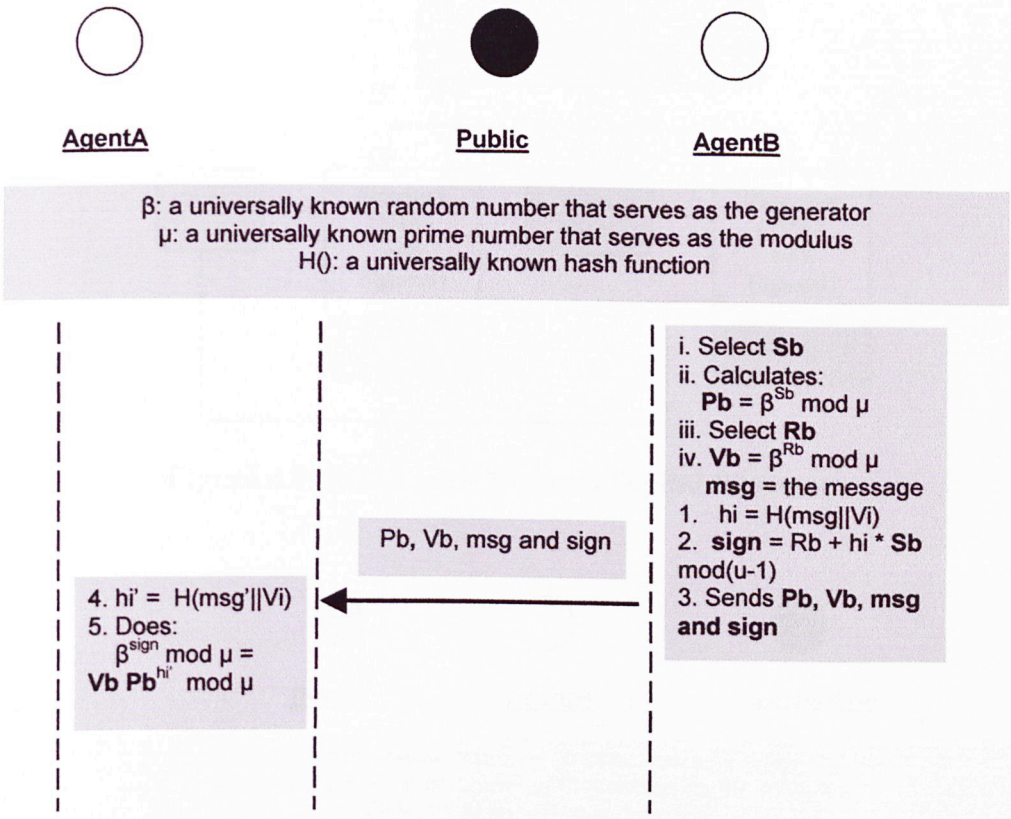

Figure 4.7: Overview of Elgamal Digital Signature

As mentioned before, H(VID) is produced by SHA1 algorithm. H(VID) is hashed/digested from attributes of the to be verified agent including date and time it was created. This H(VID) is used along with Elgamal Digital Signature to ensure the process of verification is truly secured. Even though a signature seems similar to a message digest, they have very different purposes in the type of protection they provide.

In fact, algorithms such as "SHA1WithRSA" use the message digest "SHA1" to initially "compress" the large data sets into a more manageable form, then sign the resulting 20 byte message digest with the "RSA" algorithm (Java ™ Cryptography Architecture (JCA) Reference Guide for Java™ Platform Standard Edition 6, 2006). In agent verification protocol, "SHA1withElgamal" used as shown in Figure 4.8 and Figure 2.12 in previous chapters. Figure 4.9 shows New and Existing Agent Verification.

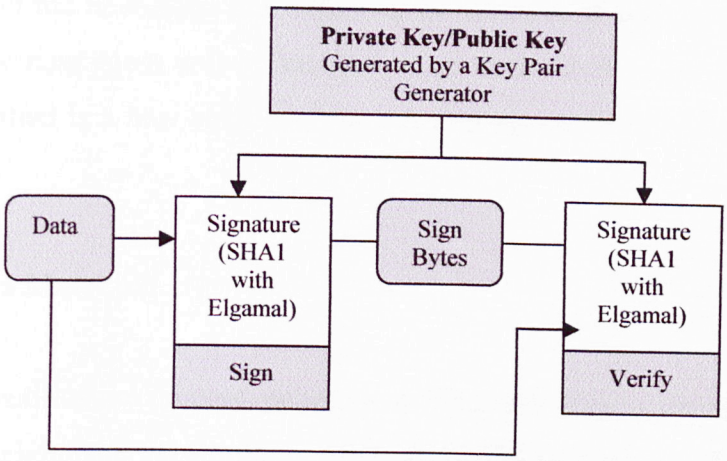

Figure 4.8: SHA1 with Elgamal Digital Signature

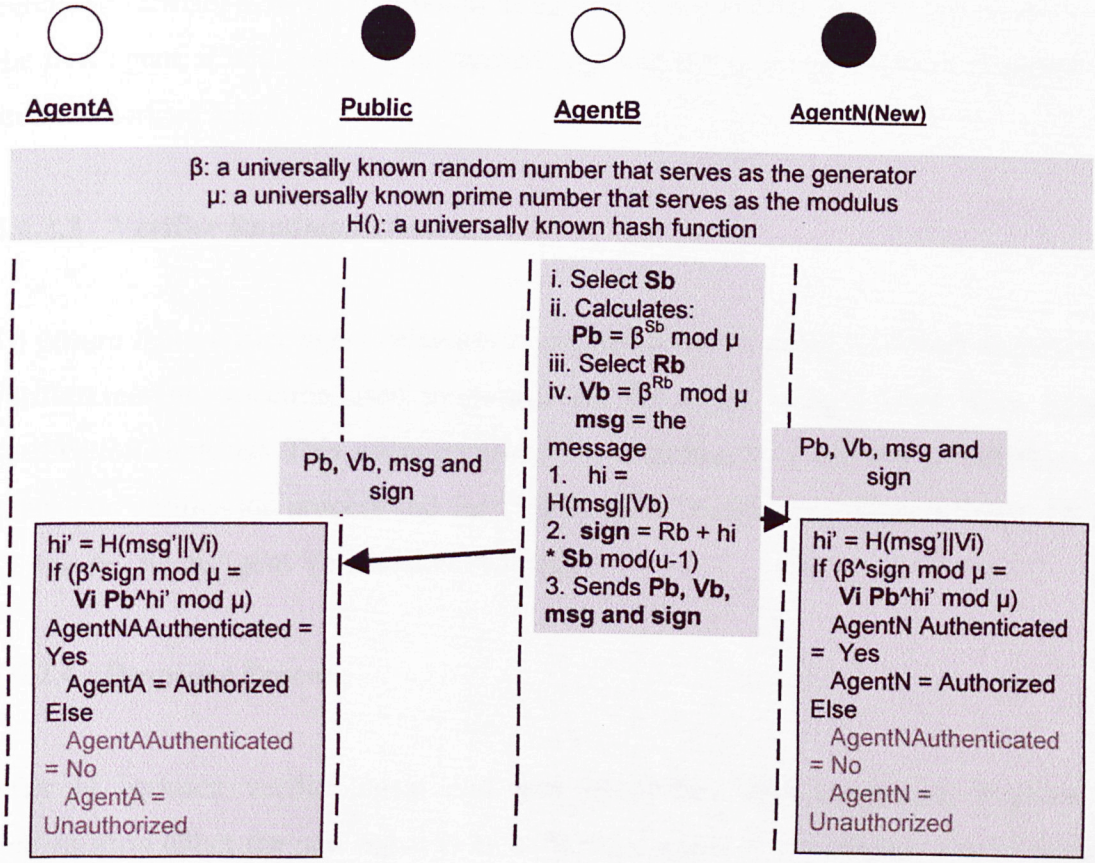

Figure 4.9: New and Existing Agent Verification

Based on Figure 4.9 above, message is sent to new agent to verify that the new agent is an authorized agent. As above mentioned, the message contains H(VID) is a hashed verification ID which is being hashed along with Vi to be hi. The H(VID) is compared

Response Interface and broadcasts it. This process is applied to existing agents as requested agent. Response is defined as $mversp_i$ in Agent Verification Protocol.

4.6.2.5 Time-based Processing

In ensuring reliability of this system is guaranteed, the process of agent verification will be launched in time-based processing. Random algorithm is used in choosing which agent will launch the verification process. The period between processes are identified as thirty minutes which means that the process will be launched at every thirty minutes. Time is defined as t_i in (R, d_i, t_i) in Agent Verification Protocol and included along with other information to be sent. t_i used to ensure that verification process commits within the required duration to be achieved. If the verification response is not received by verifier agent or exceeds the time limit, verifier agent will declare that the new agent failed to response for verification. Therefore, the new agent will be assigned as unauthorized agent.

4.6.2.6 Agent Database (DB)

Agent Verification is connected with AgentDB in identifying all information of each agent in system are stored in this database. The attribute of agent is hashed to get H(VID) as described in Agent Security section. Contents of the AgentDB are shown in Database Design in section 3.11.

4.6.3. Algorithm Description

In designing Agent Verification Algorithm, the researcher assumes that agents connected with each other by P2P connection and message queue procedure is followed. Besides that verifier random selection is done. Agent Verification Algorithm has two functions which are verifier() and verified(). Firstly, B and μ generated for both functions as public random and prime numbers. Function verifier() begins with condition ($t == t_S$) for verification process which means that verifier() is only able to launch when time for S is similar with current time and if not failure(t == false) will be returned. Condition *for* used to ensure only one process of encryption

to the H(VID) at the new agent and sign will be verified. If both of the H(VID) are similar, then the new agent will be verified as an authorized agent. In this case, the agent to be verified is a new agent and for existing agents to be verified, it used the same algorithm.

4.6.2.2 Pattern Matching

In doing agent verification process, pattern matching technique is a part of the process which is used to compare the similarity of H(VID). The contents of H(VID) has been described in section 3.3. If the pattern of H(VID) in the hi at verifier is similar to H'(VID) in the hi at the new agent, the new agent will be assigned as an authorized agent. Otherwise, if H(VID) in the hi at verifier is not similar to H'(VID) in the hi at the new agent, it will resulting unverified sign and the new agent will be assigned as an unauthorized agent.

4.6.2.3 Verifier Random Selection

To ensure that verifier agent selection is unpredictable in doing verification process, verifier random selection used to protect verifier agent being known when agent verification launched. This selection process is important to avoid any possibilities of attacks on verification process and the verifier agent. Verifier selection defined as S in (S, m_R, d_S, t_S) in Agent Verification Protocol.

4.6.2.4 Response Process

After hi between verifier agent and new agent has been compared, it gives a confirmation either the new agent is an authorized agent or unauthorized agent based on sign verification. If the new agent has been verified as an authorized agent, the verifier agent will receives a verification message from the new agent which assigned that the new agent is an authorized agent and send it to other agents. Otherwise, if the verifier agent not received the verification message, it will kill connection with the new agent and send alert warning verification response to other agents and do the same procedure. Verification Response will trigger an alarm trough Verification

occurred each time. Only when i = 0 to i +1, function will be launched or else failure(i > 0) will be returned. S uses B and μ to compute Pb and Va. Then, msg[i] created which is including (R, m_R, d_S , t_S) which represents (Receiver, message to R, description of S, time of S). After that, msg[i] being hashed along with Vb resulting hi ← H(msg[i] || Vb). S computes sign with generates Rb + hi * Sb mod μ - 1. Finally, verifier() returns Pb, Vb, msg, sign.

In other side, function verified() receives (Pb, Vb, msg, sign) as input to verify sign received. Before that, condition if used to ensure that Pb, Vb, msg and sign are truly received or else failure(Pb, Vb, msg, sign == false) will be returned. Then, function verified() hashed msg[i] along with Vb declared as H'(msg[i] || Vb) resulting hi' to be used in sign verification. Then, sign will be verified by comparing β^{sign} mod μ with Vb $Pb^{hi'}$ mod μ. If results of both is similar then function verified() will return $mveri_R$. Before that, condition *for* used to ensure only one process of sign verification is occurred each time. Only when i = 0 to i +1, msg[i] will be decrypted or else failure(i > 0) will be returned.

After sign verification process is done in function verified(), function verifier() will receive verification message $mveri_R$ for verification response process. The algorithm for verification response only begins with condition if ($t <= t_R$) which means that verifier() is only able to launch verification process when time for R is less than current time and if not, failure($t > t_R$) will be returned. This condition used to ensure that verifier agent receives verification message $mveri_R$ in specific time limit, otherwise function verification response verifiresp() will be called. If this condition is committed, condition if ($mveri_R$ == true) is created to ensure that the verification message $mveri_R$ truly received by verifier agent. Finally, if ($mveri_R$ == true) is committed, verifier agent will call function log(R, d_R , t_R) which is defined as (Receiver, description of R, time of R) to logs the verification process into verification log. After that, function $mveri_S$ = (i, d_S , t_S) is called which is defined as (other agent, description of S, time of S) to sends verification message to other agents. Otherwise, if ($mveri_R$ == true) is not committed, function verification response verifiresp() will be called.

Agent Verification Algorithm

Generates B, μ

```
verifier()
{
    If (t == tS)
    {
        for(i = 0 to i + 1)
        {
            Computes Pb ← β^Sb mod μ
            Computes Vb ← β^Rb mod μ
            Create msg[i] ← (R, mR, dS , tS)
            Computes hi ← H(msg[i] || Vb)
            Computes sign ← Rb + hi * Sb mod μ - 1
            i++
            return Pb, Vb, msg, sign
        } return failure(i > 0)
    } return failure(t == false)

    if(t <= tR)
    {
        if(mveriR == true)
        {
            log(R, dR , tR)
            mveriS = (i, dS , tS)
        } else
        {
            verifiresp()
        }
    }else
    {
        verifiresp()
    }
}

verified(Pb, Vb, msg, sign)
{
    If(Pb, Vb, msg, sign == true)
    {
        for(i = 0 to i + 1)
        {
            Computes hi' ← H'(msg[i] || Vb)
            Create msg[i] ← (S, mR, dS , tS)
            verify (sign)
            if(β^sign mod μ == Vb Pb^hi' mod μ)
            {
                return mveriR
            }
        }return failure(i > 0)
    }return failure(Pb, Vb, msg, sign == false)
}
```

Figure 4.10: Agent Verification Algorithm (continues)

```
verifiresp()
{
        kill(R, dR , tR)
        log(R, dR , tR)
        mverspS = (i, dR,tR)
}
```

Figure 4.10: Agent Verification Algorithm (continued)

As mentioned above, verification response procedure in Agent Verification called function verifiresp() because of function ($t <= t_R$) and ($mveri_R$ == true) are not being committed. Function verification response verifiresp() will kill connection with the verified agent using function kill(R, d_R , t_R) which defined as (Receiver, description of R, time of R) and calls function log(R, d_R , t_R) to logs the verification response into verification log. Finally, verification response message being sent to other agents using function $mversp_S$ = (i, d_R,t_R) defined as (other agent, description of R, time of R) to do same procedure by calling function verifiresp(). Agent Verification Algorithm is shown in Figure 4.10.