Your Project Title

An industrial training project report submitted

to

MANIPAL UNIVERSITY

For Partial Fulfillment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Information Technology

by

Your Name

Reg. No. 160953xyz

Under the guidance of

Name of Guide/Supervisor

Designation of Guide

Name of Company/Organization

MANIPAL INSTITUTE OF TECHNOLOGY

A Constituent Institute of Manipal University, Manipal

Month Year

CERTIFICATE

This is to certify that **Mr./Ms XYZ** (Reg.No. 140953xyz), a student of x year B.Tech (Computer and Communications) from Manipal Institute of Technology has completed his/her internship with us from DD-MM-YY to DD-MM-YY.

During this tenure he/she has worked on project titled "**Title of your project**" and has successfully completed it to the best of his/her abilities. His/her conduct and attendance has been good during this tenure.

ACKNOWLEDGEMENTS

My sincere thanks to XYZ.

ABSTRACT

My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract. My abstract.My abstract.

[**Security and Privacy**]: Cryptographykey management, symmetric cryptography; Security Servicesauthentication, access control

Contents

List of Tables

List of Figures

ABBREVIATIONS

LDA	:	Latent Drichlet Allocation
API	:	Application Programming Interface

NOTATIONS

α	:	Smoothing factor for words
β	:	Smoothing factor for topics

Chapter 1

Chapter Title

1.1 Section 1

Electronic reference is given in [1]. Journal article is given in [2]. Article from conference is given in [3]. Material from book [4]. Manual detail is given in [5]. Detail of technical report is given in [6]. A master thesis [7] has been referred. A PhD thesis [8] has been referred. Last referenec is taken from [9].

Definition 1 *vvhffff*

Definition 2 *fffffff*

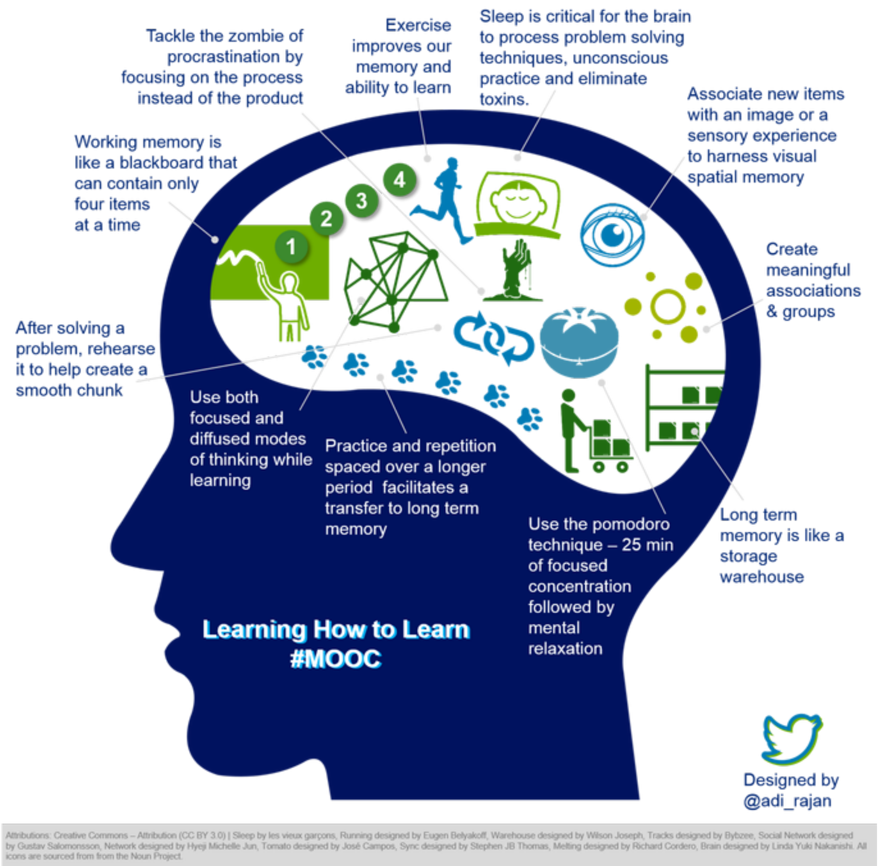

Figure 1.1: Learning how to learn

Chapter 2

Chapter Title

2.1 Section 1

Definition 3 *vvvvv*

Definition 4 *ttttttt*

Chapter 3

Chapter Title

3.1 ffffh

Chapter 4

Chapter Title

4.1 abvvv

Chapter 5

Chapter Title

5.1 ghf

Chapter 6

Chapter Title

6.1 vvvvvv

Chapter 7

Conclusion

7.1 fff

Appendices

Appendix A

Code (if required)

A.1 Kerberos Protocol

```
MODULE main

VAR

--Creating agents which are to type agtype
agA : agtype;
agB : agtype;
agS : agtype;
agI : agtype;
Iactive: boolean;

--Assigning initial to values to all variables

ASSIGN
init(agA.state):=wait;
init(agB.state):=wait;
init(agS.state):=wait;
init(agI.state):=wait;
init(agA.count):=0;
init(agB.count):=0;
init(agS.count):=0;
init(agI.count):=0;
init(agA.authenticated):=FALSE;
init(agB.authenticated):=FALSE;
init(agI.authenticated):=FALSE;
init(agS.authenticated):=TRUE;
```

```
--Transitions for the variable indicating presence or absence of intruder

next(Iactive):=
case
!Iactive:{0,1};
Iactive & agI.state=receive4beta:{0,1};
1:Iactive;
esac;

--Transitions for agent A's state

next(agA.state):=
case
agA.state=wait: send1;
agS.state=send2 & agA.state=send1: receive2;
agA.state=receive2: send3alpha;
agB.state=send4alpha & agA.state=send3alpha: receive4alpha;
agA.state=receive4alpha: wait;
1:agA.state;
esac;

--Transitions for agent B's state

next(agB.state):=
case
agA.state=send3alpha & agB.state=wait: receive3alpha;
agI.state=send3beta & agB.state=wait & Iactive: receive3beta;
agI.state=send3beta & agB.state=send4alpha & Iactive: receive3beta;
agB.state=receive3alpha:send4alpha;
agB.state=receive3beta:send4beta;
agB.state=send4alpha:wait;
1:agB.state;
esac;

--Transitions for Server S's state

next(agS.state):=
case
agA.state=send1 & agS.state=wait: receive1;
agS.state=receive1:send2;
agS.state=send2:wait;
1:agS.state;
esac;
```

```
--Transitions for the Intruder's state

next(agI.state):=
case
agI.state=wait & agA.state=send3alpha & agB.state=wait & Iactive: receive3beta;
agI.state=receive3beta & Iactive: send3beta;
agI.state=send3beta & agB.state=send4beta & Iactive  : receive4beta;
agI.state=receive4beta & Iactive: wait;
1:agI.state;
esac;

--Transitions for Agent A's counter

next(agA.count):=
case
agA.state=send1|agA.state=receive2: agA.count;
agA.state=send3alpha & agA.count<1:agA.count+1;
agA.count=1 & agA.state=receive2: 0;
1:agA.count;
esac;

--Transitions for Agent B's counter

next(agB.count):=
case
agB.state=receive3beta & agB.count<2|agB.state=receive3alpha & agB.count<2: agB.count+1;
agB.state=send4alpha |agB.state=send4beta:agB.count;
agB.count=1 & agA.state=receive4alpha & !Iactive:0;
agB.count=2 & agA.state=send3alpha|agB.count=1 & agA.state=send3alpha: 0;
1:agB.count;
esac;

--Transitions for Agent I's counter

next(agI.count):=
case
agI.state=receive3beta & agI.count<2 & Iactive:agI.count+1;
agI.state=send3beta & Iactive:agI.count;
agI.state=receive4beta & agI.count<2 & Iactive: agI.count+1;
agI.count=2: 0;
1:agI.count;
esac;

--Transitions for variable indicating agent A's authentication
```

```
next(agA.authenticated):=
case
agA.state=receive4alpha :TRUE;
1:agA.authenticated;
esac;

--Transitions for variable indicating agent B's authentication

next(agB.authenticated):=
case
agB.state=send4alpha |agB.state=send4beta :TRUE;
1:agB.authenticated;
esac;

--Transitions for variable indicating agent B's authentication which
--indicates that it has received the fourth message

next(agI.authenticated):=
case
agI.state=receive4beta :TRUE;
1:agI.authenticated;
esac;

--Agent S always is authenticated so transitions to the false state do not occur

next(agS.authenticated):=
case
1:agS.authenticated;
esac;

--Specifications which detect the presence of replay attack

--Agent B should not receive more messages than what agent A has sent it
--SPEC AG!(agA.count < agB.count);

--Agent I should never receive the fourth message
SPEC AG!(agI.state=receive4beta);

--Module for each agent's type which includes the agent's state variable,
--its counter and its authentication variable

MODULE agtype
```

```
VAR
state: {wait, send1, receive1, send2,receive2,
send3alpha, send3beta, receive3alpha, receive3beta,
send4alpha,send4beta, receive4alpha, receive4beta };
count:{0,1,2};
authenticated:boolean;
```

A.2 Kerberos Protocol with Freshness Concept

```
MODULE main

VAR

--Creating agents which are to type agtype
agA : agtype;
agB : agtype;
agS : agtype;
agI : agtype;
Iactive: boolean;
Fresh:0..20;
Time:0..20;

--Assigning initial to values to all variables

ASSIGN
init(agA.state):=wait;
init(agB.state):=wait;
init(agS.state):=wait;
init(agI.state):=wait;
init(agA.count):=0;
init(agB.count):=0;
init(agS.count):=0;
init(agI.count):=0;
init(agA.authenticated):=FALSE;
init(agB.authenticated):=FALSE;
init(agI.authenticated):=FALSE;
init(agS.authenticated):=TRUE;
init(Fresh):=0;
init(Time):=0;
```

```
--Transitions for the variable indicating presence or absence of intruder

next(Iactive):=
case
!Iactive:{0,1};
Iactive & agI.state=receive4beta:{0,1};
1:Iactive;
esac;

--Transitions for agent A's state

next(agA.state):=
case
agA.state=wait: send1;
agS.state=send2 & agA.state=send1: receive2;
agA.state=receive2: send3alpha;
agB.state=send4alpha & agA.state=send3alpha: receive4alpha;
agA.state=receive4alpha: wait;
1:agA.state;
esac;

--Transitions for agent B's state

next(agB.state):=
case
agA.state=send3alpha & agB.state=wait & Fresh=0: receive3alpha;
agI.state=send3beta & agB.state=wait & Iactive & Fresh=0: receive3beta;
agI.state=send3beta & agB.state=send4alpha & Iactive & Fresh=0: receive3beta;
agB.state=receive3alpha:send4alpha;
agB.state=receive3beta:send4beta;
agB.state=send4alpha:wait;
1:agB.state;
esac;

--Transitions for Server S's state

next(agS.state):=
case
agA.state=send1 & agS.state=wait: receive1;
agS.state=receive1:send2;
agS.state=send2:wait;
1:agS.state;
esac;
```

```
--Transitions for the Intruder's state

next(agI.state):=
case
agI.state=wait & agA.state=send3alpha & agB.state=wait & Iactive: receive3beta;
agI.state=receive3beta & Iactive: send3beta;
agI.state=send3beta & agB.state=send4beta & Iactive  : receive4beta;
agI.state=receive4beta & Iactive: wait;
agI.state=send3beta & Time>2: wait;
1:agI.state;
esac;

--Transitions for Agent A's counter

next(agA.count):=
case
agA.state=send1|agA.state=receive2: agA.count;
agA.state=send3alpha & agA.count<1:agA.count+1;
agA.count=1 & agA.state=receive2: 0;
1:agA.count;
esac;

--Transitions for Agent B's counter

next(agB.count):=
case
agB.state=receive3beta & agB.count<2|agB.state=receive3alpha & agB.count<2: agB.count+1;
agB.state=send4alpha|agB.state=send4beta:agB.count;
agB.count=1 & agA.state=receive4alpha & !Iactive:0;
agB.count=2 & agA.state=send3alpha|agB.count=1 & agA.state=send3alpha: 0;
1:agB.count;
esac;

--Transitions for Agent I's counter

next(agI.count):=
case
agI.state=receive3beta & agI.count<2 & Iactive:agI.count+1;
agI.state=send3beta & Iactive:agI.count;
agI.state=receive4beta & agI.count<2 & Iactive: agI.count+1;
agI.count=2: 0;
1:agI.count;
esac;
```

```
--Transitions for variable indicating agent A's authentication

next(agA.authenticated):=
case
agA.state=receive4alpha :TRUE;
1:agA.authenticated;
esac;

--Transitions for variable indicating agent B's authentication

next(agB.authenticated):=
case
agB.state=send4alpha|agB.state=send4beta :TRUE;
1:agB.authenticated;
esac;

--Transitions for variable indicating agent B's authentication which
--indicates that it has received the fourth message

next(agI.authenticated):=
case
agI.state=receive4beta :TRUE;
1:agI.authenticated;
esac;

--Agent S always is authenticated so transitions to the false state do not occur

next(agS.authenticated):=
case
1:agS.authenticated;
esac;

--Transitions for the freshness variable

next(Fresh):=
case
agA.state=send3alpha & agB.state=wait:0;
Fresh<20:Fresh+1;
1:0;
esac;

--Transitions for the Intruder's timer variable

next(Time):=
```

```
case
agI.state=receive3beta:0;
Time<20:Time+1;
1:0;
esac;

--Specifications which detect the presence of replay attack

--Agent B should not receive more messages than what agent A has sent it
SPEC AG!(agA.count < agB.count);

--Agent I should never receive the fourth message
SPEC AG!(agI.state=receive4beta);

--Module for each agent's type which includes the agent's state variable,
--its counter and its authentication variable

MODULE agtype

VAR
state:{wait, send1, receive1, send2,receive2,
send3alpha, send3beta, receive3alpha, receive3beta,
send4alpha, send4beta, receive4alpha, receive4beta};
count:{0,1,2};
authenticated:boolean;
```

Appendix B

Trace Files

B.1 Replay Attack

```
-- specification AG !(agA.count < agB.count)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  agA.state = wait
  agA.count = 0
  agA.authenticated = 0
  agB.state = wait
  agB.count = 0
  agB.authenticated = 0
  agS.state = wait
  agS.count = 0
  agS.authenticated = 1
  agI.state = wait
  agI.count = 0
  agI.authenticated = 0
  Iactive = 0
-> Input: 1.2 <-
-> State: 1.2 <-
  agA.state = send1
  agS.count = 2
-> Input: 1.3 <-
-> State: 1.3 <-
  agS.state = receive1
-> Input: 1.4 <-
-> State: 1.4 <-
```

```
  agS.state = send2
-> Input: 1.5 <-
-> State: 1.5 <-
  agA.state = receive2
  agS.state = wait
-> Input: 1.6 <-
-> State: 1.6 <-
  agA.state = send3alpha
  Iactive = 1
-> Input: 1.7 <-
-> State: 1.7 <-
  agA.count = 1
  agB.state = receive3alpha
  agI.state = receive3beta
-> Input: 1.8 <-
-> State: 1.8 <-
  agB.state = send4alpha
  agB.count = 1
  agI.state = send3beta
  agI.count = 1
-> Input: 1.9 <-
-> State: 1.9 <-
  agA.state = receive4alpha
  agB.state = receive3beta
  agB.authenticated = 1
-> Input: 1.10 <-
-> State: 1.10 <-
  agA.state = wait
  agA.authenticated = 1
  agB.state = send4beta
  agB.count = 2
-- specification AG !(agI.state = receive4beta)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  agA.state = wait
  agA.count = 0
  agA.authenticated = 0
  agB.state = wait
  agB.count = 0
  agB.authenticated = 0
  agS.state = wait
  agS.count = 0
```

```
  agS.authenticated = 1
  agI.state = wait
  agI.count = 0
  agI.authenticated = 0
  Iactive = 0
-> Input: 2.2 <-
-> State: 2.2 <-
  agA.state = send1
  agS.count = 2
-> Input: 2.3 <-
-> State: 2.3 <-
  agS.state = receive1
-> Input: 2.4 <-
-> State: 2.4 <-
  agS.state = send2
-> Input: 2.5 <-
-> State: 2.5 <-
  agA.state = receive2
  agS.state = wait
-> Input: 2.6 <-
-> State: 2.6 <-
  agA.state = send3alpha
  Iactive = 1
-> Input: 2.7 <-
-> State: 2.7 <-
  agA.count = 1
  agB.state = receive3alpha
  agI.state = receive3beta
-> Input: 2.8 <-
-> State: 2.8 <-
  agB.state = send4alpha
  agB.count = 1
  agI.state = send3beta
  agI.count = 1
-> Input: 2.9 <-
-> State: 2.9 <-
  agA.state = receive4alpha
  agB.state = receive3beta
  agB.authenticated = 1
-> Input: 2.10 <-
-> State: 2.10 <-
  agA.state = wait
  agA.authenticated = 1
  agB.state = send4beta
```

```
  agB.count = 2
-> Input: 2.11 <-
-> State: 2.11 <-
  agA.state = send1
  agI.state = receive4beta
```

B.2 Replay Attack overcome using Freshness Concept

```
-- specification AG !(agA.count < agB.count)  is true
-- specification AG !(agI.state = receive4beta)  is true
```

References

[1] M. Shell. (2007) IEEEtran webpage on CTAN. [Online]. Available: http://www.ctan.org/tex-archive/macros/latex/contrib/IEEEtran/

[2] Y. Okada, K. Dejima, and T. Ohishi, "Analysis and comparison of PM synchronous motor and induction motor type magnetic bearings," *IEEE Trans. on EE*, vol. 31, pp. 1047–1053, Sep./Oct. 1995.

[3] R. K. Gupta and S. D. Senturia, "Pull-in time dynamics as a measure of absolute pressure," in *Proc. IEEE International Workshop on Microelectromechanical Systems (MEMS'97)*, Nagoya, Japan, Jan. 1997, pp. 290–294.

[4] B. D. Cullity, *Introduction to Magnetic Materials.* Reading, MA: Addison-Wesley, 1972.

[5] *FLEXChip Signal Processor (MC68175/D)*, Motorola, 1996.

[6] R. Jain, K. K. Ramakrishnan, and D. M. Chiu, "Congestion avoidance in computer networks with a connectionless network layer," Digital Equipment Corporation, MA, Tech. Rep. DEC-TR-506, Aug 1987.

[7] A. Karnik, "Performance of TCP congestion control with rate feedback: TCP/ABR and rate adaptive TCP/IP," M. Eng. thesis, Indian Institute of Science, Bangalore, India, jan 1999.

[8] Q. Li, “Delay characterization and performance control of wide-area networks,” Ph.D. dissertation, Univ. of Delaware, Newark, May 2000. [Online]. Available: http://www.ece.udel.edu/~qli

[9] L. Roberts, “Enhanced proportional rate control algorithm PRCA,” ATM Forum Contribution 94-0735R1, Aug. 1994.