Investigation of Network Topology Poisoning Attacks in Software Defined Networking

A Graduate Project Report submitted to Manipal University in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Electronics and Communication Engineering

Submitted by

Garuda Suma Pranavi

Reg. No.: 130907120

Under the guidance of

Prof. Ma Maode
Associate Professor
Department of Electrical and Electronics Engineering
Nanyang Technological University

&

Prof. Stanley Oswald Maben
Associate Professor
Department of Electronics and Communication Engineering
Manipal Institute Of Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MANIPAL INSTITUTE OF TECHNOLOGY

(A Constituent College of Manipal University)

MANIPAL – 576104, KARNATAKA, INDIA

JANUARY-JUNE 2017

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
MANIPAL INSTITUTE OF TECHNOLOGY
(A Constituent College of Manipal University)
MANIPAL – 576 104 (KARNATAKA), INDIA

Manipal
27/06/2017

CERTIFICATE

This is to certify that the project titled **Investigation Of Network Topology Poisoning Attacks in Software Defined Networking** is a record of the bonafide work done by **Garuda Suma Pranavi**(*Reg. No. 130907120*) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (BTech) in **ELECTRONICS AND COMMUNICATION ENGINEERING** of Manipal Institute of Technology Manipal, Karnataka, (A Constituent College of Manipal University), during the academic year 2016 - 2017.

Prof. Stanley Oswald Maben
Project Guide

Prof. Dr. M. Sathish Kumar
HOD, E & C.
M.I.T, MANIPAL

Manipal Institute Of Technology, Karnataka
03/07/2017

NANYANG TECHNOLOGICAL UNIVERSITY

Certificate of Completion

This certificate is awarded to

GARUDA SUMA PRANAVI

in recognition of her participation in the

NTU-India Connect Research Internship Programme 2017

with **Assoc Prof Ma Maode,**

School of Electrical & Electronic Engineering

25/05/2017

Date

Professor B.V.R. Chowdari
Senior Executive Director
President's Office

ACKNOWLEDGEMENTS

"…the beauty of the destination is half veiled and the fragrance of success half dull, until the traces of all those enlightening the path are left to fly with the wind spreading words of thankfulness…"

First and foremost, I would like to express my sincerest gratitude to my Project guide, Prof. Ma Maode of the Department of Electrical and Electronic Engineering at Nanyang Technological University for his continuous support on my research, for his patience, motivation, enthusiasm and immense knowledge. The door to Prof. Ma's office was always open whenever I ran into a trouble spot or had a question about my research. He consistently allowed this project to be my own work, but steered me in the right direction whenever he thought I needed it. I could not have imagined a better advisor and mentor for enlightening me the first glance of research.

I would like to acknowledge, Prof. Stanley Oswald Maben as the internal guide to my thesis who I'm gratefully indebted to, for his valuable suggestions and help throughout the project.

I would also like to thank Dr. Gopalakrishna Prabhu, Dr. M. Sathish Kumar, Dr. Somashekara Bhat, Dr. Bore Gowda, Mr. Ravilla Dilli, Prof. Padma Shri Ma'am without whose recommendations and support I wouldn't have had the opportunity to work in such a prestigious institution for my Bachelor's thesis.

I thank my fellow partners working under Prof. Ma, Qiu Yue and Kiruthika Saravanan for the stimulating discussions and the sleepless nights we were working together. I would also like to convey my heartfelt gratitude to the NTU-India Connect Team and Ms. Tay Ting Fang for giving me this unique opportunity and helping me in each step of the process.

ABSTRACT

Software Defined Networking, a new networking paradigm has transformed the way enterprises design, build and operate networks. Network innovations in SDN are faster as they are more open and easier to program which allows operators more control, customization and optimization. Since the technology is still immature, there are a lot of software vulnerabilities. Hence, SDN security needs to be built into the architecture, as well as delivered as a service to protect the availability, integrity, and privacy of all connected resources and information in a network. Our aim is to design effective solutions (in the form of algorithms) against these malicious attacks.

The first two months of the internship were dedicated for literature survey on all security issues in SDN. Then, I had to select one possible attack and work on a possible countermeasure (through formal verification tools). The algorithm is first written in a mathematical descriptive language called BAN where we try to resolve the problems it has in reaching the goals. A more comprehensive analysis is performed on Scyther, which performs automated analysis of security protocols. To make the algorithm more efficient in terms of time taken by the protocol, we first find time taken by individual protocols in C/C++ and then calculate the time taken by the security protocol (which is a combination of other protocols) while under continuous known/unknown attack through MATLAB.

After the extensive literature survey, I decided to work on improving the AuthFlow protocol (Authentication and Access Control Mechanism for SDN based on host credentials). This protocol still follows MS-CHAP v2 which has been proven to be prone to man-in-the-middle and ARP injection attacks among others, and thus we try to make it more secure. However, if both the server and client associated themselves with certificates, we can adopt EAP-TLS as suggested in the future work, the security and efficiency of which have been discussed in detail. Incase of large administrator overheads, where it is not possible to have certificates on both sides of communication, we have also analyzed and made more efficient EAP-TTLS and PEAP, all of which can be employed in the AuthFlow mechanism.

Once, we adopt the secure version of MS-CHAPv2 or EAP-TLS, EAP-TTLS and PEAP we can define access control according to the privilege level of each host, by using the host identity as a new flow field to define forwarding rules and to deny access to hosts without valid credentials. A combination of BAN logic, Scyther, C/C++ and MATLAB was used in the project. This is a very important step in defining security for SDN as it becomes more open and programmable as a part of the larger cloud infrastructure.

LIST OF FIGURES

Contents

CHAPTER 1
INTRODUCTION

As SDN transforms networks into an open and programmable component of the larger cloud infrastructure, the technology faces lot more security threats than the traditional networks.

1.1. Motivation:

The AuthFlow paper used as reference employs MS-CHAP v2 with LDAP for verifying host credentials which has been proven to be very weak for authentication due to it its vulnerability to ASLEAP and ARP-injection attacks. Due to the rapid spread of SDN, we must make the employed algorithms more secure. Current research focusses on scripting security modules and just guessing the shortcomings, we have moved a step further and employed a unique research/mathematical analysis for SDN security through the use of BAN, Scyther, C/C++ and MATLAB.

1.2 Objective:

Taking into consideration the administrator and economic constraints, I have worked on improving a group of protocols: MS-CHAPV2, EAP-TLS, EAP-TTLS and PEAP which can be used according to the availability of certificates. The security and efficiency of these protocols have been worked on in detail.

1.3 Project Work Schedule:

The first 2 months had been a comprehensive literature survey on SDN and security protocols adopted. April'17 was dedicated for proving the security of protocols through BAN and their automated analysis through Scyther. In May, I learnt various security protocols such as DES, MD-4, SHA-1 etc. and implemented them in C/C++ so as to calculate time taken by each algorithm. In June,I had to had make MS-CHAPv2, EAP-TLS, EAP-TTLS and PEAP more efficient through the use of individual security protocols, which was analysed in MATLAB.

1.4 Organization of Report:

Chapter 1: Introduction
This chapter gives a brief on why SDN security should be worked upon, the motivation behind the project, the objective and project work schedule.

Chapter 2: Background Theory/ Literature Review
This chapter will be a brief insight into the most innovative and unique solutions proposed in SDN uptil now. These include the attacks most commonly deployed against SDN and the solutions found to overcome them.

Chapter 3: Methodology
This chapter specifies the algorithms involved, their mathematical and automated analysis which identifies their shortcomings and then makes it more secure and more efficient (proven through MATLAB).

Chapter 4: Result analysis
This chapter will be a description of the results obtained through implementation and simulation.

Chapter 5: Work conclusion and Future scope of SDN and the discussed proposals.

CHAPTER 2
BACKGROUND THEORY

Summarized in this chapter are the most innovative and unique solutions explored during my literature review. In later chapters, I have improved upon techniques to be used for authentication.

2.1. Overview of SDN

Software Defined Networking emerged as an attempt to introduce network innovations faster and to radically simplify and automate the management of large networks. In this emerging network paradigm, the control and management of the network is separated from the traffic forwarding primitives. The centralized control plane monitors the whole network and makes decisions on packet forwarding for the switches (data plane). The interface to the switches is OpenFlow which provides network administrators with a simple and uniform abstraction to the configuration of different physical or virtual multi vendor's network devices. Specifically, the SDN controller inserts and updates traffic rules for the current traffic flows into one or more flow tables inside each switch. The resulting decoupling simplifies network monitoring, fault tolerance, security policy enforcement but it also introduces new security issues. In this report we discuss the security implications data plane programmability brings about.

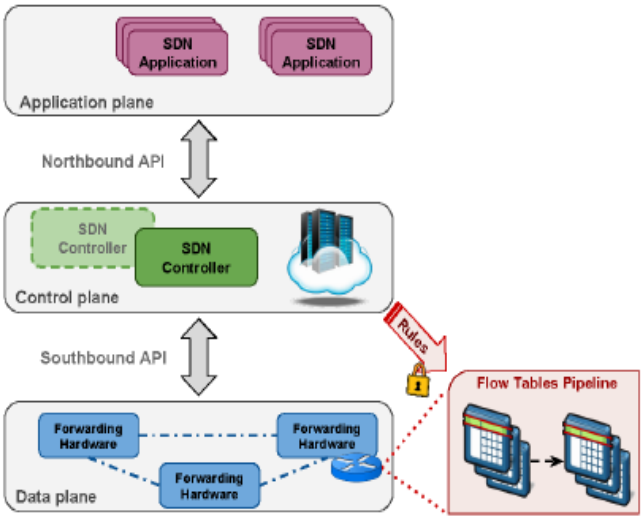


Fig. 1

The Northbound API is exposed to developers so as to develop simple-to-use high level abstractions devised to hide the complexity inherent in the underlying network topology and release the network administrator from the need to deal with low level network nodes configuration details. They basically allow the programmability of SDN. However, not only can they be used to develop security applications that use the network equipment as a check point for compliance, they can also constitute a privileged entering point for attackers to introduce malicious applications. Through the Southbound interface, network devices can be controlled, configured, managed and monitored by the controller so as to change their forwarding behaviour and adapt to changing business requirements and to make the network more responsive to real time traffic demands. For securing the southbound access, authentication is important. Other basic SDN review involved learning about match/action abstraction, platforms and enabling technologies such as OpenState, SDPA, programming languages such as P4, Domino, SNAP and event driven programming.

2.2. Denial of Service (DoS) attacks: The basic idea of DoS attacks is to overload network links or to flood a victim with a massive number of flows until it fails to serve legitimate users. Generally, there are two modes in which rules can be installed at switches: proactive and reactive. For the proactive mode, the controller first breaks down network policies into flow rules, and installs them at switches when the network bootstraps. For the reactive mode, the controller computes and installs rules only when a switch explicitly requests them. Clearly, reactive mode enables switches to quickly adapt to network dynamics and does not require switches to have large flow tables. However, the reactive rule installation also makes SDN controllers and switches vulnerable to denial of service (DoS) threats. It has 3 main subcategories:

2.2.1. Control channel congestion attack: Even though switches individually maintain their control channels with the controller, these logically separate channels may share some common physical links and hence, flow requests flooded by the victim switch overwhelm common links, and normal flow requests using those links experience congestion. In addition, if the packet buffer of the victim switch sends entire packets instead of just packet headers to the controller, it results in even higher bandwidth consumption.

- Simple port throttling mechanisms such as Floodlight have fixed thresholds which block all flows from a certain MAC address/port punishing benign hosts. Moreover, they cannot adapt to network dynamics. Hence, security enhanced floodlight was introduced which

includes a digitally authenticated Northbound API, where the administrator is asked to sign the OpenFlow application class or a set of actions before they start new flows or modify old ones.

- Additive Increase Multiplicative Decrease is a better mechanism as it allows dynamic adjusting of the requesting rates of switches rather than temporarily blocking.

- Resource Management through trust values allows for priority assignment while placing it in the buffer queue instead of the first come first serve mechanism. It also decides timeouts for the flows according to the current traffic in the network once it checks the number of requests from the concerned producer host.

- Single Layer Fair Queuing polls the requests from each switch according to a weighted round robin approach, but hosts under the same switch as the attacker are punished unfairly. Hence, the Multi-Layer Fair Queuing approach was proposed where initially each queue corresponds to a group of switches which is dynamically expanded to per-switch and maybe per-port queues according to the requirement and these can also be aggregated back.

*2.2.2. Controller Resource Saturation Attack***:** If the flooded flow requests arrive at the controller, they will consume the controller's resource (CPU, memory, bandwidth etc.) for rule computation and installation. Without any protection, the controller's resource can be saturated by the flooded requests and legitimate requests may be dropped.

- Avant-Guard was proposed which implements a SYN proxy but it may cause a buffer saturation attack. The LineSwitch (deployed on edge switches) overcomes this problem by proxying only the new flows to remove the possibility of SYN flooding with spoofed source IP addresses by blacklisting them.

- The FloodGuard uses the proactive flow rule analyzer and the packet migration tool to forward all table-miss packets to data plane cache. Using the real time rate of packet_in messages, the flooding attack is identified against a certain anomaly threshold.

- The controller protection protocol includes a proof of work, so that even the attacker needs to dedicate a large amount of computational resources in order to send large amounts of attack traffic, making an attack potentially prohibitively expensive.

- The Flow Ranger combines already proposed approaches of trust value assignment (with the help of ISP's) for each packet_in message and for priority queues. Moreover, the message scheduling is performed according to a weighted round robin strategy.

2.2.3. Flow table overflow attacks: DoS attacks purposely create a large number of new flows that extend and update the forwarding tables of the switches. To provide functionality even when resources or communication bandwidth are over consumed, several methods were proposed.

- One aspect is rate limiting which can be employed in various ways: by limiting the amount of traffic that a single port can send to the switch, limiting the amount of packets sent to the controller, or limiting the number of rules that the controller can insert into the switch in a short period of time.

- Flow timeouts can be adjusted, with smaller time-to-live properties, thus making flow tables difficult to be overflowed.

- FLIP (Fast Lightweight Policy preserving SDN Updates) combines rule replacements and additions to avoid scenarios such as blackholes and evil loops.

- *A* peer support strategy is proposed when a switch runs out of memory space where other switches support the targeted switch by offering their idle flow table resources to install new rules and mitigate the attack.

- UDP flooding can be identified by pre-evaluating thresholds, assigning the variable state as a flooder and dropping packets if necessary. Traffic percentage triggers can be employed by comparing the outgoing traffic to the rest of the network traffic, where this threshold must not be exceeded. Amplification rate triggers can be used to keep a check on the DNS, NTP and some specific ports.

2.3. Firewalls: They prevent unauthorized access to or from a private network. SDN can employ the centralized or distributed firewall approach. The port knocking process allows a stateful firewall to grant access to one or multiple hosts on a specific port only if they are able to successfully conclude a pre-defined sequence of connection attempts (moving target defence technology). Such ordered sequence represents a shared secret between the firewall and the hosts. However, sniffing, sequence replay attacks, port scans can identify this sequence.

- Advanced port knocking authentication schemes with QRC using AES was proposed where the key is shared through SMS generating an OTP which is verified through timestamps and then the sequence is encrypted through AES and compared to the generated sequence at the servers, allowing or denying access to the client by opening the port.

- Covert communication through steganography, where the encrypted sequence is hidden in the LSB's of an image selected by the sender and sent as the payload of packets used for knocking the ports providing secrecy through obscurity.

- Methods where a pre-configured text passphrase is verified after the successful port knock sequence is used (to overcome NAT attacks) in the Secure Port Knock Tunnelling mechanism to increase the protection of the authentication process.

- Port hopping mechanisms are also used, but synchronization is very important for these implementations. Future work can be based on port sequence selection.

2.4. Stateful Failure recovery (controller independent stateful link/node failure detection and recovery scheme).Restoration and protection are the most common failure recovery strategies employed. In the restoration strategy, alternate paths are installed in the flow tables after failure detection and network resources are dynamically allocated which delays the recovery process. However, the protection strategy has alternate backup paths preconfigured in the network.

- The 1:1 path protection scheme tags packets with MPLS labels upon detecting a node or link failure in order to communicate to previous nodes to use a detour path for subsequent packets (rerouting through the fast failover feature).But inconsistency and latency can be imposed by forging these tags in the absence of encryption and authentication protocols.

- Backup resources can be computed depending on the importance level of the link- the no. of backup paths or taking the reactive recovery strategy, where the backup path is identified by VLAN tagging instead of IP matching for high speed and low memory requirements.

- Multi Topology Routing based IP fast re-route: Multiple routing configurations provide a self-recovering SDN against single failures in the data plane by central computation of virtual topologies isolating single nodes.

2.5 Malicious application mitigation mechanisms/ Authentication mechanisms:

- AuthFlow is a credential based authentication and access control mechanism which uses the host identity as a new flow field to define forwarding rules and allow different levels of access to network resources according to the privilege level of each host.

- LegoSDN was proposed to increase controller reliability in the case of SDN application failures, by proposing an isolation layer between SDN applications which support atomic updates, efficient rollbacks and a fault tolerance layer to overcome crash triggering events.

- ROSEMARY uses a micro NOS architecture for each OpenFlow application effectively sandboxing the application to protect the control layer from any vulnerability or malicious operation of the application.

- FRESCO facilitates the development of security applications through API's for scripting, reusable models and a database for storage and management of session information. It defines forward, drop and group actions each using redirect, mirror and quarantine.

- FortNox allows for efficient detection and reconciliation of potentially conflicting flow rules imposed by the dynamic OpenFlow applications through priority assignment. (OF Operator, OF Security and OF Application)

2.6 Other Security proposals:

- Anomaly detection mechanisms such as SPHINX use a trustworthy topology graph to detect anomalous messages from switches. This uses table dependency graphs to raise alerts when it detects untrusted entities triggering changes to existing flow behaviour.

- A malicious user can try to abuse the DNS messages in order to bypass the access policies and send data. DNS attacks can be prevented by keeping a track of all the resolved IP addresses through an array based global variable.

- MTD (Moving Target Defense) technologies change network parameters irregularly over time to increase the attacker's cost. It can be used for randomizing TCP/UDP ports, IP addresses and network paths to improve network resiliency.

- Flow management module decides timeouts and routing paths for each flow according to threat probability through intrusion detection systems. Monitoring modules collect multiple statistics about flows, switches and links such as flow throughput, switch TCAM usage and link bandwidth usage which helps in redirecting malicious traffic through the path having least utilized links.

- History based approaches calculate hash values for linearized critical events and then use a distributed verification approach such as in NetCo which propose reliable routing through unreliable routers.

2.7 Security proposal selected for further analysis and improvement:

AuthFlow is an authentication and access control mechanism based on host credentials developed to ensure the proper operation of SDN. The main contributions are:

1. A credential based authentication to perform an access control according to the privilege level of each host, through mapping the host credentials to the relevant set of flows.
2. A host authentication mechanism just above the MAC layer in an OpenFlow network, which guarantees a low overhead and ensures a fine-grained access control.
3. A new framework for control applications, enabling Software Defined Network controllers to use the host identity as a new flow field to define forwarding rules.

The proposed mechanism applies the IEEE 802.1X standard and Extensible Authentication Protocol (EAP). EAP encapsulates authentication messages exchanged between the supplicant host and RADIUS authentication server. The translation of IEEE 802.1X messages into RADIUS packets is performed by the authenticator. The AuthFlow application allows or denies network traffic from a host depending on the result of authentication between the host and authenticator.

When a host compatible with IEEE 802.1X starts, it also starts the authentication phase by sending a start message to a reserved multicast MAC address (01:80:C2:00:00:03) with ethernet type 0x888E. The authentication procedure does not depend on any host's prior knowledge about the network, nor on a translation of an IP address into a MAC address (avoiding address spoofing attacks) as it works just above the MAC layer.

The IEEE 802.1X packets are forwarded directly to the authenticator. The authenticator is a RADIUS client that implements 802.1X and forwards the content of EAP messages to RADIUS. The authenticator sends a confirmation message of success for POX (in this case) over a secure,

encrypted and authenticated channel using SSL 3.0 and Public Key Infrastructure (PKI). As EAP allows the use of several different authentication methods, the method adopted was MS-CHAP v2 (Microsoft Challenge-Handshake Authentication Protocol) which authenticates virtual routers against a database using username and password as credentials using a Lightweight Directory Access Protocol (LDAP). It can also store other parameters that can define the privileges of the router for network access.

The AuthFlow authentication mechanism works as follows: A virtual router sends an authentication request, standardized by IEEE 802.1X, and the controller redirects it to the authenticator. Authenticator responds to it and the host sends its credentials. The authenticator checks the credentials of the supplicant against RADIUS server, running the authentication method defined in EAP. If the authentication procedure succeeds, the authenticator sends a success message to the host and a confirmation message for the controller through a SSL channel. This message identifies the supplicant by its MAC address, confirms the success of authentication and also informs the identity of the supplicant host. After authentication, our AuthFlow application allows the host to access network resources. In case of revocation of authentication, the authenticator communicates with the AuthFlow application which immediately denies the host access to the network, erasing and blocking all flow entries from and to the banned host.

The implementation (MS-CHAPv2 authentication with a LDAP database) is considered weak in the realm of authentication and is extensible to other authentication mehtods, such as EAP-TLS or its modified versions which authenticates hosts based on certificates for access credentials.

2.8 Security Proof through BAN (Burrows- Abadhi-Needham Logic)

BAN is a formal logic to prove the logic correctness and find vulnerabilities in the modelling of a given network protocol. BAN can be used to prove security properties of network protocols without explicitly reasoning about an intruder's actions while only reasoning about the actions of honest roles under unknown environmental conditions.Below, we briefly review the original BAN logic with an emphasis on notation, idealization, and inference rules relevant to our research.

- $P \mid\equiv X$: P believes X
- $P \lhd X$: P sees X
- $P \mid\sim X$: P once said X
- $P \Rightarrow X$: P has jurisdiction over X
- $\#(X)$: X is fresh
- $\{X\}_k$: X is encrypted with k
- $P \xleftrightarrow{K} Q$: P and Q share a secret key K

The extensions for describing a certificate proposed are as follows:

- $\wp\mu(P, K_P)$: P has a good public key K_p
- $\wp\gamma(K_P^{-1})$: P has a good private key K_p^{-1}
- $\delta(X, K_P^{-1})$: X is signed with P's private key K_p^{-1}
- $(\theta(t_1^P, t_2^P), X)$:X holds good in the time interval (t1,t2)
- $\Omega(t_1^P, t_2^P)$:(t1,t2) is a good time interval
- $Cert_P$:Certificate of P issued by C

2.8.1 Logic Postulates

R1. Message meaning rule:

$$\frac{P \models (P \xleftrightarrow{K} Q), \quad P \triangleleft \{X\}_K}{P \models (Q \triangleleft X)}$$

If P and Q share key K and P sees X encrypted under K, then P believes Q once said X.

R2. Nonce-verification rule:

$$\frac{P \models (\#(X)), \quad P \models (Q \triangleleft X)}{P \models (Q \models X)}$$

If P believes that X is fresh and Q has seen X, then P believes that Q believes X.

R3. Jurisdiction rule:

$$\frac{P \models (Q \Rightarrow X), \quad P \models (Q \models X)}{P \models X}$$

If an agent P believes that Q controls certain information, and P believes that Q believes that information, then P believes that information.

Extended BAN Inference rules:

R4. Message meaning (for public key) rule:

$$\frac{P \models \wp\mu(Q, K_Q), P \models \wp\gamma(Q, K_Q^{-1}), P \triangleleft \delta(X, K_Q^{-1})}{P \models (Q \triangleleft X)}$$

If Q has a good public and private key and X is signed by Q's private key, then Q has seen X.

R5. The see-signed message rule:

$$\frac{P \triangleleft \delta(X, K_Q^{-1})}{P \triangleleft X}$$

If P sees X signed with the private key of Q, then it must have seen X also.

R6. The certificate duration stamp rule:

$$\frac{P \models Q \vdash (\theta(t_1^P, t_2^P), X), P \models Q \vdash \Omega(t_1^P, t_2^P)}{P \models Q \models X}$$

If Q says that X holds true in (t1,t2) and (t1,t2) is a good time interval, then Q will believe in X.

CHAPTER 3
METHODOLOGY

3.1 Flowchart

Firstly, we model the protocol mathematically through BAN logic and try to reach it's goals. However, when we cannot reach the required goals through the logic postulates, we conclude that the protocol is nto secure as in the case of MS-CHAP v2 with LDAP.

The second is proof/ improvement through the results obtained from Scyther analysis which includes detailed scipting of all messages exchanged in the protocol (with its encryption and hashing functions implemented). The sole purpose of including Scyther codes in the report is to explain how the protocol works and what aspects of security are checked in the analysis.

The third step was implementing individual security functions employed in the protocol in C/C++ and calculate time taken by the protocol (as they vary with different processor's and RAM). Functions such as DES, SHA-1, SHA-256, HMACSHA-256, MD4, MD5 and RSA have been implemented. Values obtained from the C/C++ analysis are obtained in the MATLAB analysis.

In the efficiency analysis through MATLAB, we calculate time taken by the protocol under continuous known/unknown attacks. Here, we write in brief the time taken by each step of the protocol through the use of encryption/hashing functions, and then we bombard them with

attacks. In case of known attacks, we know the secure protocol proceeds to completion. However, in the case of unknown attacks, we are not that sure (Known/ unknown attacks generated through random generator functions). Hence, we use another random generator under unknown attacks which specifies which step on the protocol is under attack and calculate time taken till that step (unsuccessful completion of the protocol) and then an average is taken for a large number of known/unknown attacks generated. This analysis on an increasing number of unknown attacks is then plotted to see if there is any considerable improvement in efficiency when our new algorithm is employed as compared to the original protocol.

3.2 MS-CHAP v2 (Microsoft Challenge Handshake Authentication Protocol Version 2)

MS-CHAP v2 was proposed to provide security for remote access connections. The process of authentication is described below in brief, followed by its descriptive representation:

- The new client/ application first requests a login challenge from the authentication server.
- The remote access server sends a challenge message to the remote access client that consists of a session identifier and a 16-byte arbitrary challenge string.
- The remote access client sends a response message that contains the user name, a 16 byte random peer authenticator challenge, a 8 byte challenge by hashing the received challenge, peer authenticator challenge and the client's username, a 24 byte reply by encrypting the 8 byte challenge with the MD-4 hashed version of the client's password.
- The server then uses the hashes of the client's password stored in the LDAP database, to decrypt the replies. If the decrypted blocks match the challenge, the client is authenticated. The server then sends an indication of the success or failure of the connection attempt, by using the 16 byte peer authenticator challenge as well the client's hashed password.
- The remote access client also computes the authenticator response and if the computed response matches the received response, mutual authentication is successfully completed and the connection is used, else it is terminated.

3.2.1 Mschapv2 original protocol in Scyther

```
hashfunction h1;
hashfunction h2;
usertype string;
const p,x, clientsuccess, serversuccess, accesslevel: string;
```

```
const magicservertoclientsigningconstant, padtomakeitdomorethanoneiteration: string;

macro m1= h1(ni, nr, x);
macro m2= h2(p);
macro m3= {m1}m2;
macro m4= h2(h2(p));
macro m5= magicservertoclientsigningconstant;
macro m6= h1(m4,m3,m5);
macro m7= padtomakeitdomorethanoneiteration;
macro m8= h1(m6, m1,m7);

protocol mschap(I,R)
{
role I
{
fresh ni: Nonce;
var nr: Nonce;

recv_1 (R, I, nr);
send_2 (I, R, ni, x, m3);
recv_3 (R, I, {clientsuccess}pk(I));
recv_4 (R, I, m8);
match (h1(h1(h2(h2(p)), m3, m5), m1,m7), m8);
send_5 (I, R, {serversuccess}pk(R));
recv_6 (R, I ,{accesslevel}pk(I));
}

role R
{
fresh nr: Nonce;
var ni: Nonce;

send_1 (R, I, nr);
recv_2 (I, R, ni, x, m3);
match (({h1(ni, nr, x)}h2(p)), m3);
send_3 (R, I, {clientsuccess}pk(I));
send_4 (R, I, m8);
```

recv_5 (I, R, {serversuccess}pk(R));
send_6 (R, I ,{accesslevel}pk(I));
}}

Fig. 2
"Scyther analysis of MSCHAP v2 original protocol"

3.2.2 MS-CHAP V2 modified protocol in Scyther

hashfunction h1;
hashfunction h2;
hashfunction h3;

```
usertype string;
const p,x, clientsuccess, serversuccess, accesslevel: string;
const magicservertoclientsigningconstant, padtomakeitdomorethanoneiteration: string;

macro m1= h1(ni, nr, x);
macro m2= h2(p);
macro me= {h2(p)}nr;
macro m3= {m1}me;
macro m4= h3(m1,m2,m3);

protocol mschap(I,R)
{
role I
{
fresh ni: Nonce;
var nr: Nonce;

recv_1 (R, I, {{nr}pk(I)}sk(R));
send_2 (I, R, {{ni, x}pk(R)}sk(I), m3);
recv_3 (R, I, {{clientsuccess}pk(I)}sk(R));
recv_4 (R, I, m4);
match (h1(h1 (ni, nr, x),h2(p), m3), m4);
send_5 (I, R, {{serversuccess}pk(R)}sk(I));
recv_6 (R, I ,{{accesslevel}pk(I)}sk(R));

claim_i1 (I, Secret, ni);
claim_i2 (I, Secret, nr);
claim_i3 (I, Secret, p);
claim_i4 (I, Secret, m1);
claim_i5 (I, Secret, m2);
claim_i6 (I, Niagree);
claim_i7 (I, Nisynch);
}

role R
{
fresh nr: Nonce;
```

```
var ni: Nonce;

send_1 (R, I, {{nr}pk(I)}sk(R));
recv_2 (I, R, {{ni, x}pk(R)}sk(I), m3);
match (({h1(ni, nr, x)}({h2(p)}nr)), m3);
send_3 (R, I, {{clientsuccess}pk(I)}sk(R));
send_4 (R, I, m4);
recv_5 (I, R, {{serversuccess}pk(R)}sk(I));
send_6 (R, I ,{{accesslevel}pk(I)}sk(R));

claim_r1 (R, Secret, ni);
claim_r2 (R, Secret, nr);
claim_r3 (R, Secret, p);
claim_r4 (R, Secret, m1);
claim_r5 (R, Secret, m2);
claim_r6 (R, Niagree);
claim_r7 (R, Nisynch);
}}
```

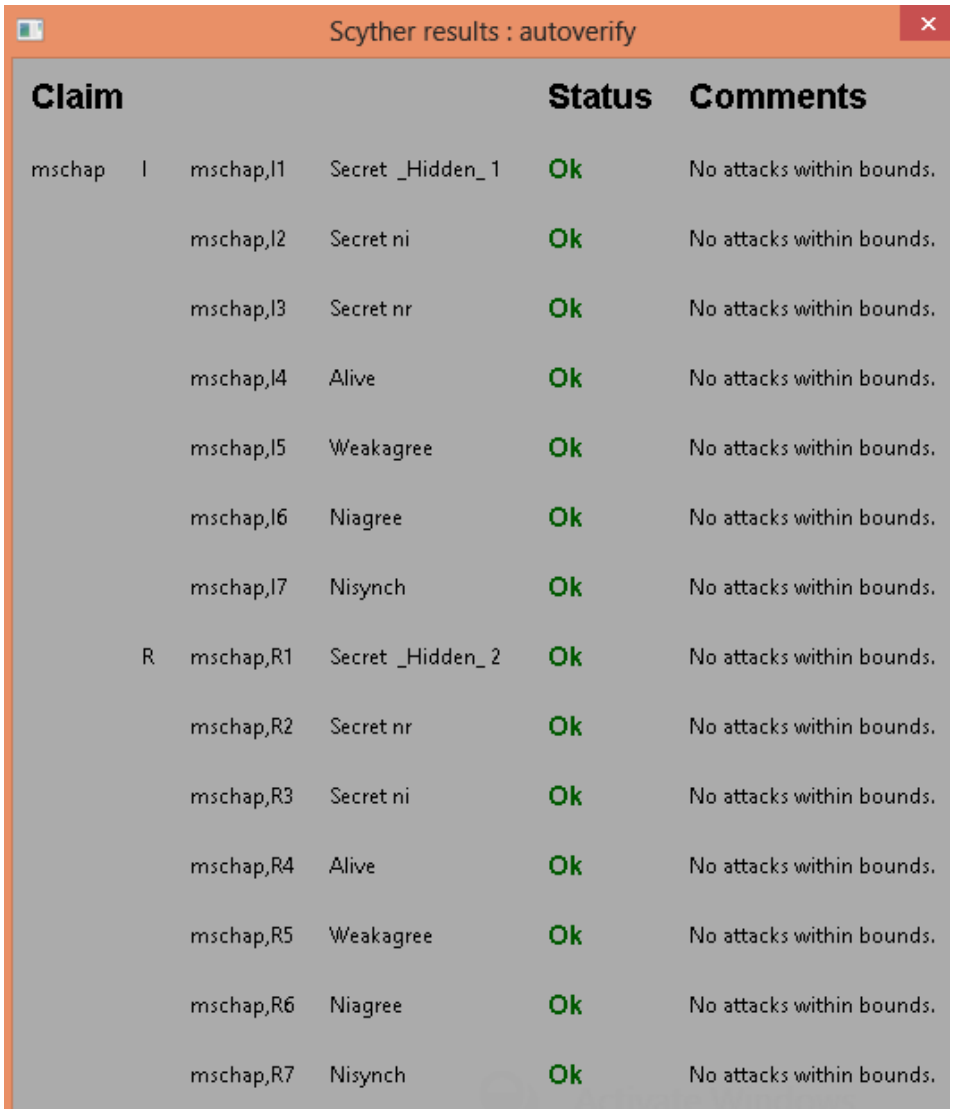
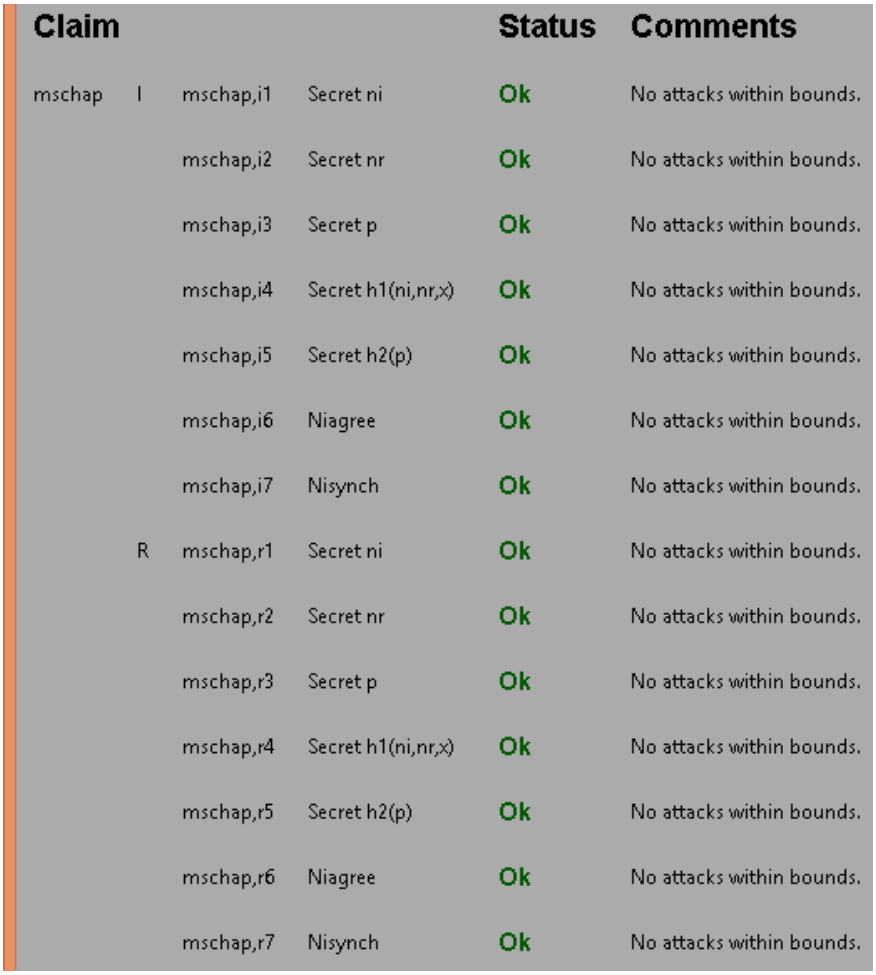


Fig. 3
"Scyther Analysis of MSCHAP v2 using claims specified by us"



Claim				Status	Comments
mschap	I	mschap,i1	Secret ni	Ok	No attacks within bounds.
		mschap,i2	Secret nr	Ok	No attacks within bounds.
		mschap,i3	Secret p	Ok	No attacks within bounds.
		mschap,i4	Secret h1(ni,nr,x)	Ok	No attacks within bounds.
		mschap,i5	Secret h2(p)	Ok	No attacks within bounds.
		mschap,i6	Niagree	Ok	No attacks within bounds.
		mschap,i7	Nisynch	Ok	No attacks within bounds.
	R	mschap,r1	Secret ni	Ok	No attacks within bounds.
		mschap,r2	Secret nr	Ok	No attacks within bounds.
		mschap,r3	Secret p	Ok	No attacks within bounds.
		mschap,r4	Secret h1(ni,nr,x)	Ok	No attacks within bounds.
		mschap,r5	Secret h2(p)	Ok	No attacks within bounds.
		mschap,r6	Niagree	Ok	No attacks within bounds.
		mschap,r7	Nisynch	Ok	No attacks within bounds.

Fig.4
"Scyther Analysis of MSCHAP v2 modified protocol using Autoverify"

3.2.3 Time analysis in MATLAB (Codes for each algorithm are specified in annexure)

```
tsha1= 0.0384; %sha1
tmd4= 0.0476; %md4
tsha256= 0.05;
tdese= 0.29165 %desencryption
tdesd= 0.29165 %desdecryption
trsae= 0.14285 %rsaencryption
trsad= 0.14285 %trsadecryption

CHAPo_a = [tsha1+tmd4+3*tdese, trsad, tmd4+tsha1+tsha1, trsae, trsad];
CHAPo_b= [tsha1+tmd4+3*tdese, trsae, tmd4+tsha1+tsha1, trsad, trsae];
```

```
CHAPo=[tsha1+tmd4+3*tdese, tsha1+tmd4+3*tdese +3*tdesd, trsae, trsad tmd4+tsha1+tsha1, tmd4+tsha1+tsha1, trsae, trsad, trsae, trsad];
for tmp = 2:length(CHAPo_a);
   CHAPo_a (tmp)= CHAPo_a (tmp-1)+ CHAPo_a (tmp);
end
for tmp = 2:length(CHAPo);
   CHAPo (tmp)= CHAPo (tmp-1)+ CHAPo (tmp);
End

CHAPm_a = [2*trsad, 2*trsae+tsha1+tmd4+tdese+3*tdese, 2*trsad, tsha256, 2*trsae, 2*trsad];
CHAPm_b= [2*trsae, 2*trsad+tsha1+tmd4+tdese+3*tdese, 2*trsae, tsha256,2*trsad, 2*trsae];
CHAPm=[2*trsae,2*trsad,2*trsae+tsha1+tmd4+tdese+3*tdese,
2*trsad+tsha1+tmd4+tdese+3*tdese, 2*trsae, 2*trsad, tsha256, tsha256,2*trsae,2*trsad, 2*trsae,
2*trsad ]
for tmp = 2:length(CHAPm_a);
   CHAPm_a (tmp)= CHAPm_a (tmp-1)+ CHAPm_a (tmp);
end
for tmp = 2:length(CHAPm);
   CHAPm (tmp)= CHAPm (tmp-1)+ CHAPm (tmp);
end
```

Original MS-CHAP v2

- tsha1+tmd4+3*tdese
- tsha1+tmd4+3*tdese+3*tdesd
- trsae+trsad
- tmd4+tsha1+tsha1
- tmd4+tsha1+tsha1
- trsae+trsad
- trsae+trsad

Modified MS-CHAP v2

- 2*trsae+2*trsad+2*trsae+tsha1+tmd4+tdese+3*tdese
- 2*trsad+tsha1+tmd4+tdese+3*tdese
- 2*trsae+2*trsad
- tsha256
- Tsha256
- 2*trsae+2*trsad
- trsae+trsad

```
total_number = 100000;
unkown_attacks= 0;
y_mschapo = zeros(1,10);          y_mschapm= zeros(1,10);
left_time_mschapo= 0;             left_time_mschapm=0;      n=1;
```

```
for x=0:0.1:0.9
   left_time_mschapo= total_number*(1-x)*CHAPo_a(length(CHAPo_a));
   left_time_mschapm= total_number*(1-x)*CHAPm_a(length(CHAPm_a));
   unknown_attacks = uint16(total_number*x);
   unexpected_delay_mschapo = randi([1,length(CHAPo_a)],1,unknown_attacks);
   unexpected_delay_mschapm = randi([1,length(CHAPm_a)],1,unknown_attacks);
   attack_total_delay_mschapo= 0;    attack_total_delay_mschapm= 0;

   for i=1:unknown_attacks
attack_total_delay_mschapo = attack_total_delay_mschapo +
CHAPo_a(unexpected_delay_mschapo(i));
attack_total_delay_mschapm=attack_total_delay_mschapm+CHAPm_a(unexpected_delay_msc
hapm(i));
   end
   y_mschapo(n)=(left_time_mschapo+attack_total_delay_mschapo)/(total_number*(1-x));
   y_mschapm(n)=(left_time_mschapm+attack_total_delay_mschapm)/(total_number*(1-x));
   n=n+1;
end
```

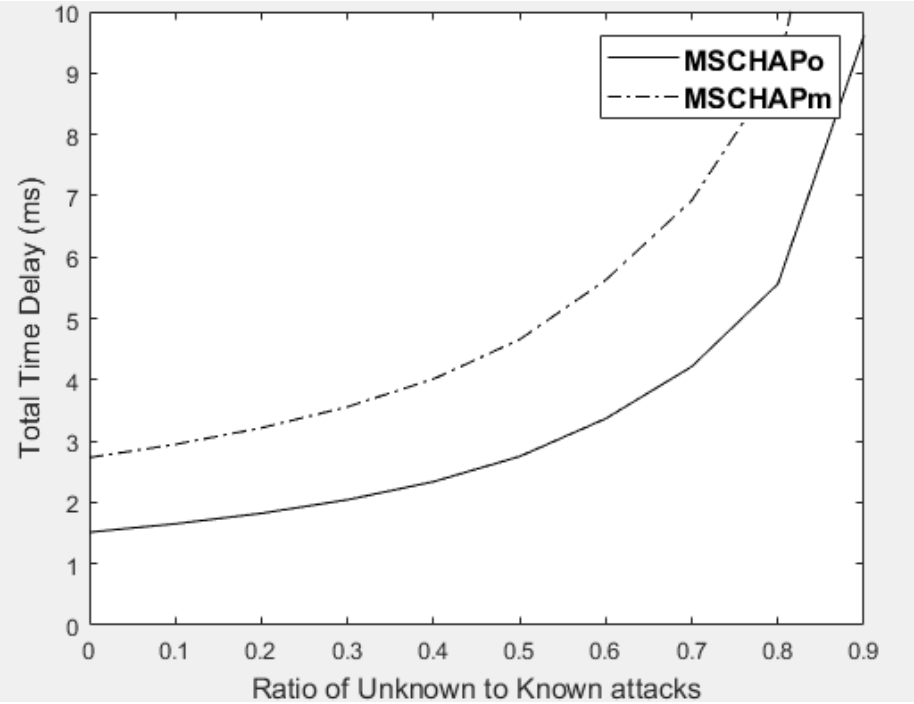


Fig. 5
"MATLAB analysis of efficiency of MS-CHAP v2"

3.3 EAP-TLS (The Extensible Authentication mechanism - Transport Layer Security protocol)

The EAP-TLS protocol, based on the Secure Sockets Layer allows applications to communicate securely through the exchange and verification of certificates. TLS specifies a framework that enables mutual authentication through symmetric or asymmetric encryption, negotiation of the specific encryption algorithm (the cipher suite) and a secured exchange of keys to be used for encrypting messages. After the authenticator and the peer negotiate EAP. The authenticator typically sends an EAP Request/Identity packet to the peer and the peer responds with its user ID in an EAP response/ identity packet. From now on, the conversation occurs between the authentication server and the peer and the authenticator just acts like a pass-through device which encapsulates the packets for transmission. The protocol proceeds as follows:

- Once the EAP server receives the peer's identity, it must respond with an EAP-TLS start packet with the start bit set and no data.
- The peer then sends an EAP Response packet, the data field of which will contain a TLS client_hello handshake message encapsulating the peer's TLS version number, a session ID, a random number and a set of ciphersuites supported by the peer.
- The EAP server responds with an EAP_Request packet which contains a TLS server_hello handshake message (with the TLS version number, another random number, as session ID and ciphersuite), TLS server_certificate (which contains unique serial number of the certificate, name of issuer, validity period of the certificate, certified public key of the server and the private key of the certifying authority signs the certificate), server_key_exchange (the public key is sent in the message to compute the premaster secret if it wasn't sent in the server_certificate), certificate_request and a server_hello_done message.
- Once the peer receives the certificate request, unless it is configured for privacy it must send the client certificate, client_key_exchange (the client computes a premaster secret using the server_key_exchange and then encrypts it with the server's public key), certificate verify, change_cipher_spec and TLS finished.
- After receiving this packet, the EAP server will verify the peer's certificate and digital signature and then send the TLS finished message. (After the client_key_exchange, we use a random ID in client hello, a random ID in server hello and the premaster secret to compute the master secret).

3.3.1 EAP-TLS Security Verification through BAN

The goals of initial authentication in the BAN Logic notation can be represented as below. The goals for the initial authentication are to exchange keying material and to achieve mutual authentication through the certificates of the host and server.

1. $P \mid\equiv Q \xleftrightarrow{PMS} P$
2. $Q \mid\equiv P \xleftrightarrow{PMS} Q$
3. $P \mid\equiv \wp\mu (K_Q, Q)$
4. $Q \mid\equiv \wp\mu (K_P, P)$

Assumptions for Initial Authentication

1. $P \mid\equiv \#(Nonce_P)$
2. $Q \mid\equiv \#(Nonce_Q)$
3. $CA \mid\equiv \xrightarrow{Kca} CA$
4. $P \mid\equiv \wp\gamma (K_{CA}^{-1})$
5. $P \mid\equiv \wp\gamma (P, K_P^{-1})$
6. $Q \mid\equiv \wp\gamma (K_{CA}^{-1})$
7. $Q \mid\equiv \wp\gamma (Q, K_Q^{-1})$
8. $P \mid\equiv Q \Longrightarrow \wp\mu (K_Q, Q)$
9. $Q \mid\equiv P \Longrightarrow \wp\mu (K_P, P)$
10. $P \mid\equiv Q \vdash \Omega (t_1, t_2)$
11. $P \mid\equiv Q \vdash (\theta (t_1, t_2), Cert_Q)$
12. $Q \mid\equiv P \vdash \Omega (t_1, t_2)$
13. $Q \mid\equiv P \vdash (\theta (t_1, t_2), Cert_P)$

Analysis

$P \triangleleft (Q \vdash (\delta(\theta(t_1, t_2), \wp\mu (K_Q, Q), K_{CA}^{-1})$
$P \mid\equiv (Q \vdash (\theta(t_1, t_2), \wp\mu (K_Q, Q))$, $P \mid\equiv Q \vdash \Omega (t_1, t_2)$
$P \mid\equiv Q \mid\equiv \wp\mu (K_Q, Q)$, $P \mid\equiv Q \Longrightarrow \wp\mu (K_Q, Q)$
$P \mid\equiv \wp\mu (K_Q, Q)$

$Q \triangleleft (P \vdash (\delta(\theta(t_1, t_2), \wp\mu (K_P, P), K_{CA}^{-1})$
$Q \mid\equiv (P \vdash (\theta(t_1, t_2), \wp\mu (K_P, P))$, $Q \mid\equiv P \vdash \Omega (t_1, t_2)$
$Q \mid\equiv P \mid\equiv \wp\mu (K_P, P)$, $Q \mid\equiv P \Longrightarrow \wp\mu (K_P, P)$
$P \mid\equiv \wp\mu (K_P, P)$

$Q \mid\equiv (P \xleftrightarrow{K_Q} Q)$, $Q \triangleleft \{PMS\}K_Q$
$Q \mid\equiv P \vdash PMS$

$Q \triangleleft \{PMS\}K_Q$, $Q \mid\equiv \wp\gamma (Q, K_Q^{-1})$
$Q \triangleleft PMS$

//Presence of a good public key implies validation of certificate and is the basis of authentication. We have also proved that Q and P both have seen the pre master secret which implies the exchange of keying material/ security parameters.

3.3.2 EAP-TLS verification on Scyther

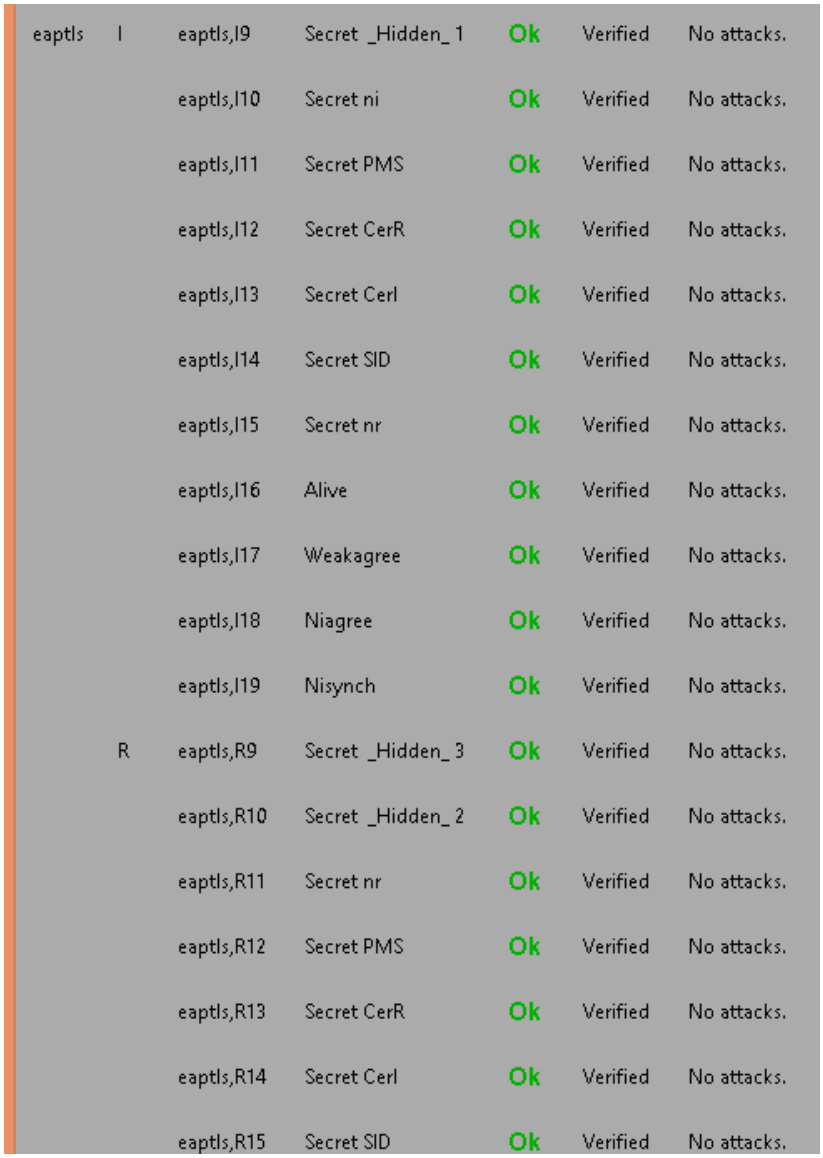
```
usertype string;
const req, shd, ccs: string;
const mastersecret, clientfinished, serverfinished, keyexpansionclient, keyexpansionserver, accesslevel, success: string;
hashfunction h, prf;

macro m=( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R));
macro m1= {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R))}sk(I);
macro ms= prf(PMS, mastersecret,ni, nr);
macro mc= ( {CerI, pk(I)}sk(CA), {PMS}pk(R));
macro m2= prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I), ccs));
macro m3= prf (ms, clientfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc)}sk(I),prf (ms, serverfinished, h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc), {h(SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, mc )}sk(I), ccs)));
macro clientkey=prf(ms, keyexpansionclient, ni, nr);
macro serverkey= prf(ms, keyexpansionserver, ni, nr);

protocol eaptls (I,R,CA)
{
role I
{
const SID, CerI, CerR,PMS: Data;
fresh ni: Nonce;
var nr:Nonce;

send_1 (I, R, SID, ni);
recv_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);
send_5 (I, CA, I);
recv_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));
send_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));
```

```
 send_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R)
)}sk(I));
recv_9 (R, I, ccs);
recv_10 (R, I,  prf(ms, serverfinished, h(m, m1)));
match (m2, prf(ms, serverfinished,  h(m, m1)));
send_11 (I, R, prf(ms, clientfinished, h(m, m1, m2)));
recv_12 (R, I, {accesslevel}serverkey);
send_13 (I, R, {success}clientkey);

claim_I1 (I, Secret, CerR);
claim_I2 (I, Secret, CerI);
claim_I3 (I, Secret, PMS);
claim_I4 (I, Secret, ms);
claim_I5 (I, Secret, clientkey);
claim_I6 (I, Secret, serverkey);
claim_I7 (I, Nisynch);
claim_I8 (I, Niagree);
}

role R
{
const SID, CerI, CerR, PMS: Data;
fresh nr: Nonce;
var ni:Nonce;

recv_1 (I, R, SID, ni);
send_2 (R, CA, R);
recv_3 (CA, R,  {R, {CerR, pk(R)}pk(R)}sk(CA));
send_4 (R, I, SID, nr, {CerR,pk(R)}sk(CA), req, shd);
recv_7 (I, R, {CerI, pk(I)}sk(CA), {PMS}pk(R));
recv_8 (I,R , {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA), {PMS}pk(R)
)}sk(I));
match(m1, {h( SID, ni, nr, {CerR,pk(R)}sk(CA), req, shd, {CerI, pk(I)}sk(CA),
{PMS}pk(R))}sk(I));
send_9 (R, I, ccs);
send_10 (R, I,  prf(ms, serverfinished, h(m, m1)));
recv_11 (I, R, prf (ms, clientfinished, h(m, m1, m2)));
```

```
match (m3, prf(ms, clientfinished, h(m, m1, m2)));
send_12 (R, I, {accesslevel}serverkey);
recv_13 (I, R, {success}clientkey);

claim_R1 (R, Secret, CerR);
claim_R2 (R, Secret, CerI);
claim_R3 (R, Secret, PMS);
claim_R4 (R, Secret, ms);
claim_R5 (R, Secret, clientkey);
claim_R6 (R, Secret, serverkey);
claim_R7 (R, Nisynch);
claim_R8 (R, Niagree);
}

role CA
{
const CerI, CerR: Data;

recv_2 (R, CA, R);
send_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));
recv_5 (I, CA, I);
send_6 (CA, I, {I, {CerI, pk(I)}pk(I)}sk(CA));

claim_CA1 (CA, Secret, CerR);
claim_CA2 (CA, Secret, CerI);
}}
```

eaptls	I	eaptls,I1	Secret CerR	Ok	Verified	No attacks.
		eaptls,I2	Secret CerI	Ok	Verified	No attacks.
		eaptls,I3	Secret PMS	Ok	Verified	No attacks.
		eaptls,I4	Secret prf(PMS,mastersecret,ni,nr)	Ok	Verified	No attacks.
		eaptls,I5	Secret prf(prf(PMS,mastersecret,ni,nr),keyexpansio...	Ok	Verified	No attacks.
		eaptls,I6	Secret prf(prf(PMS,mastersecret,ni,nr),keyexpansio...	Ok	Verified	No attacks.
		eaptls,I7	Nisynch	Ok	Verified	No attacks.
		eaptls,I8	Niagree	Ok	Verified	No attacks.
	R	eaptls,R1	Secret CerR	Ok	Verified	No attacks.
		eaptls,R2	Secret CerI	Ok	Verified	No attacks.
		eaptls,R3	Secret PMS	Ok	Verified	No attacks.
		eaptls,R4	Secret prf(PMS,mastersecret,ni,nr)	Ok	Verified	No attacks.
		eaptls,R5	Secret prf(prf(PMS,mastersecret,ni,nr),keyexpansio...	Ok	Verified	No attacks.
		eaptls,R6	Secret prf(prf(PMS,mastersecret,ni,nr),keyexpansio...	Ok	Verified	No attacks.
		eaptls,R7	Nisynch	Ok	Verified	No attacks.
		eaptls,R8	Niagree	Ok	Verified	No attacks.
	CA	eaptls,CA1	Secret CerR	Ok	Verified	No attacks.
		eaptls,CA2	Secret CerI	Ok	Verified	No attacks.

Fig.6
"Scyther Analysis of EAP-TLS according to security claims specified by us"



eaptls	I	eaptls,I9	Secret _Hidden_1	Ok	Verified	No attacks.
		eaptls,I10	Secret ni	Ok	Verified	No attacks.
		eaptls,I11	Secret PMS	Ok	Verified	No attacks.
		eaptls,I12	Secret CerR	Ok	Verified	No attacks.
		eaptls,I13	Secret CerI	Ok	Verified	No attacks.
		eaptls,I14	Secret SID	Ok	Verified	No attacks.
		eaptls,I15	Secret nr	Ok	Verified	No attacks.
		eaptls,I16	Alive	Ok	Verified	No attacks.
		eaptls,I17	Weakagree	Ok	Verified	No attacks.
		eaptls,I18	Niagree	Ok	Verified	No attacks.
		eaptls,I19	Nisynch	Ok	Verified	No attacks.
	R	eaptls,R9	Secret _Hidden_3	Ok	Verified	No attacks.
		eaptls,R10	Secret _Hidden_2	Ok	Verified	No attacks.
		eaptls,R11	Secret nr	Ok	Verified	No attacks.
		eaptls,R12	Secret PMS	Ok	Verified	No attacks.
		eaptls,R13	Secret CerR	Ok	Verified	No attacks.
		eaptls,R14	Secret CerI	Ok	Verified	No attacks.
		eaptls,R15	Secret SID	Ok	Verified	No attacks.

Fig. 7
"Scyther Analysis of EAP-TLS using Autoverify"

3.3.3 Timing Analysis in MATLAB

trsae= 0.1667 %rsaencryption
trsad= 0.1667 %rsadecryption
thmacsha256= 0.0714
tsha1= 0.0384; %sha
tsha256= 0.05 %sha256
tmd5= 0.05263; %md5
tdese= 0.29165;

tdesd= 0.29165;

EAP-TLS original

- trsae
- trsad
- tsha1+trsae
- tsha1+trsad
- 5*thmacsha256+2*thmacsha256
- 2*thmacsha256+ 2*thmacsha256
- 14*thmacsha256+tdese
- 14*thmacsha256+tdesd+tdese
- tdesd

EAP-TLS modified

- trsae
- trsad
- tsha1+trsae
- tsha1+trsad
- 3*thmacsha256+tsha256+thmacsha256
- 3*thmacsha256+tsha256+thmacsha256
- 10*thmacsha256+tdese
- 10*thmacsha256+tdesd+tdese
- tdesd

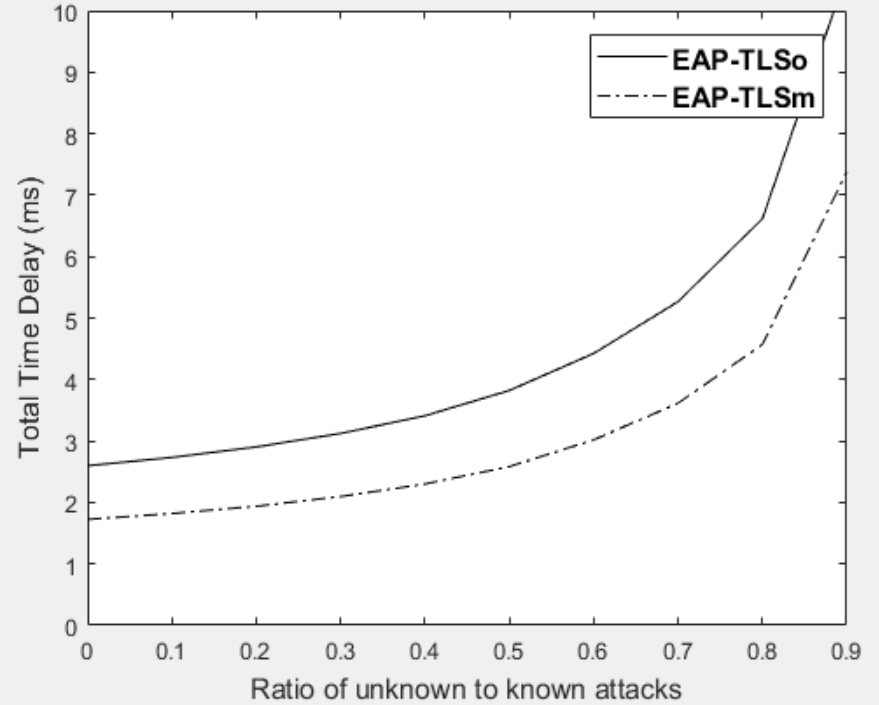


Fig.8
"MATLAB analysis of efficiency of EAP-TLS"

EAP-TLS is a superior form of EAP, it is completely password cracking resistant as it does not rely on user passwords. EAP-TLS relies on digital certificates on both the server and the client end to facilitate mutual authentication and secure key exchange. Unfortunately, the need for public key infrastructure deployment on the server end and the installed user base was a barrier for many organizations. Hence, a new IETF standard called EAP-TTLS (Tunneled Transport Layer security) was proposed to ease the deployment requirements by producing a standard that only required digital certificates on the authentication server end and not on the client end.

3.4 Extensible Authentication Protocol – Tunneled Transport Layer Security (EAP-TTLS)

EAP-TTLS is an EAP method that encapsulates a TLS session, consisting of a handshake phase and a data phase. During the handshake phase, the server is authenticated by the client (or client and server are mutually authenticated) using standard TLS procedures and keying material generated in order to create a cryptographically secure tunnel for information exchange in the subsequent data phase. During the data phase, the client is authenticated to the server, using an arbitrary authentication mechanism such as EAP, PAP, CHAP, MS-CHAP or MS-CHAP v2 encapsulated within the secure tunnel. Considering CHAP authentication in the TLS session established.

3.4.1 Challenge Handshake Authentication Protocol (CHAP)

CHAP uses PPP and extends the user authentication functionality provided on Windows networks to remote workstations (by integrating their encryption and hashing algorithms). It is used to periodically verify the identity of the peer using a 3 way handshake. The process of authentication is as follows:

- After the link establishment phase (exchange of LCP packets), the authenticator sends a "challenge" message to the peer.
- The peer responds with a value calculated using a "one way hash" function such as MD5.
- The authenticator checks the response against its own calculation of the expected hash value. If the values match, authentication is acknowledged, otherwise the connection should be terminated.
- At random intervals, the authenticator sends a new challenge to the peer and repeats the process.

Therefore, EAP allows legacy password based authentication protocols to be used against existing authentication databases, while protecting the security of these legacy protocols against eavesdropping, man-in-the-middle and other attacks. The data phase may also be used for additional arbitrary data exchange.

3.4.2 EAP-TTLS original/modified protocol verification on Scyther

```
hashfunction prf;  hashfunction h;

usertype String;
const ccs, csu, success, accesslevel, clientfinished, serverfinished: String;
const ttlschallenge, ttlskeyingmaterial,mastersecret, shd, password, uname, unamenottrue: String;
macro ms= prf (pms, mastersecret, (ni, nr));
macro finishedi=  h(ms, clientfinished, SID, ni, csu, nr,  {CerR, pk(R)}sk(CA), shd, {pms}pk(R));
macro finisheds= h(ms, serverfinished, SID, ni, csu, nr,  {CerR, pk(R)}sk(CA), shd, {pms}pk(R) , finishedi);
macro clientkey= prf (ms, ttlskeyingmaterial, (ni, nr));
macro serverkey= prf (ms, ttlskeyingmaterial, (ni, nr));
macro chapchallenge= h (ms, ttlschallenge, ni,nr);
macro mschap= (uname, chapchallenge, password);

protocol eapttls (I, R, CA)
{
role I
{
const SID, CerR, pms, S: Data;
fresh ni: Nonce;
var nr: Nonce;

send_1 (I, R, unamenottrue, SID, ni, csu);
recv_4 (R, I, SID, nr, csu, {CerR, pk(R)}sk(CA), shd);
send_5 (I, R, {pms}pk(R), ccs);
send_6 (I, R, finishedi);
recv_7 (R, I, ccs, finisheds);
match (h (ms, serverfinished, SID, ni, csu, nr,  {CerR, pk(R)}sk(CA), shd, {pms}pk(R) , finishedi), finisheds);
send_9 (I, R, {mschap}clientkey);
```

recv_10 (R, I, {success, accesslevel} serverkey);

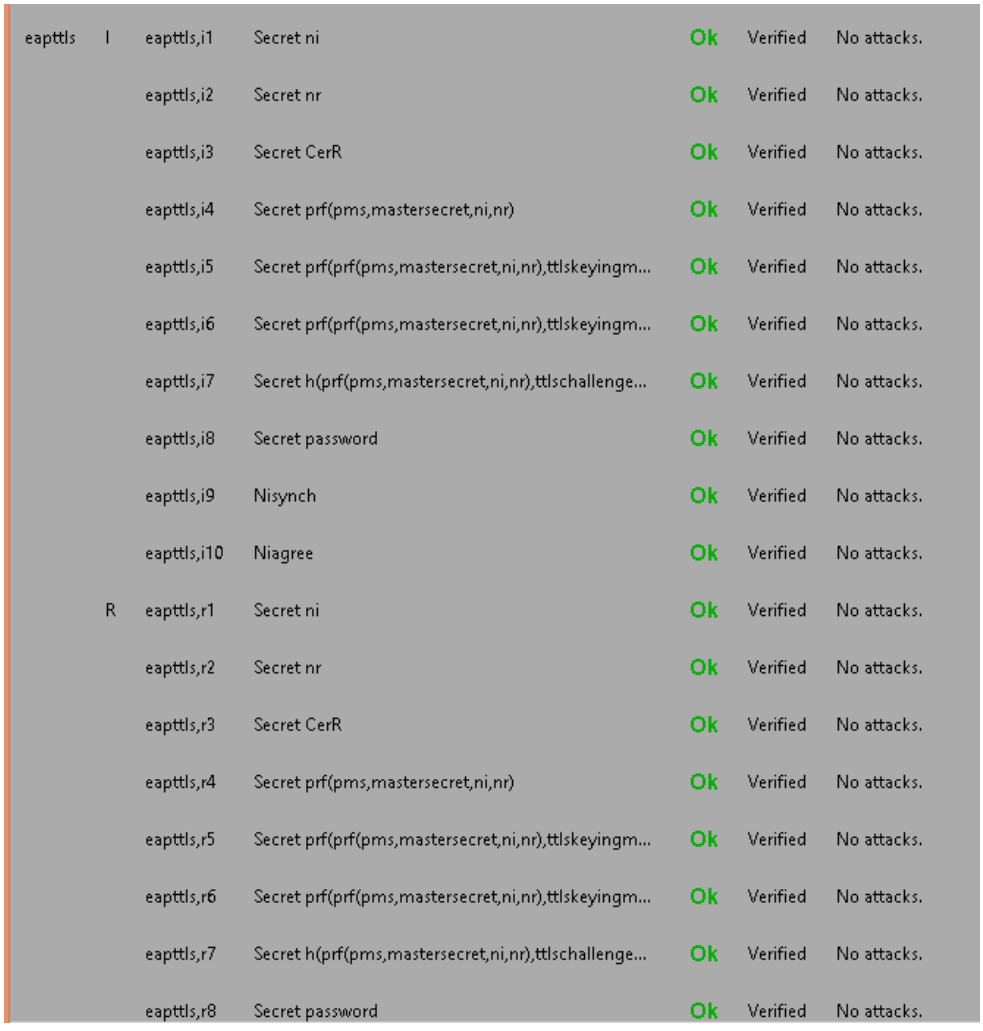
claim_i1 (I, Secret, ni);
claim_i2 (I, Secret, nr);
claim_i3 (I, Secret, CerR);
claim_i4 (I, Secret, ms);
claim_i5 (I, Secret, clientkey);
claim_i6 (I, Secret, serverkey);
claim_i7 (I, Secret, chapchallenge);
claim_i8 (I, Secret, password);
claim_i9 (I, Nisynch);
claim_i10 (I, Niagree);
}

role R
{
const SID, CerR, pms, S: Data;
fresh nr: Nonce;
var ni: Nonce;

recv_1 (I, R, unamenottrue, SID, ni, csu);
send_2 (R, CA, R);
recv_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));
send_4 (R, I, SID, nr, csu, {CerR, pk(R)}sk(CA), shd);
recv_5(I, R, {pms}pk(R), ccs);
recv_6 (I, R, finishedi);
match (h (ms, clientfinished, SID, ni, csu, nr, {CerR, pk(R)}sk(CA), shd, {pms}pk(R)), finishedi);
send_7(R, I, ccs, finisheds);
recv_9 (I, R, {mschap}clientkey);
match ((uname, h (ms, ttlschallenge, ni,nr), password), mschap);
send_10 (R, I, {success, accesslevel}serverkey);

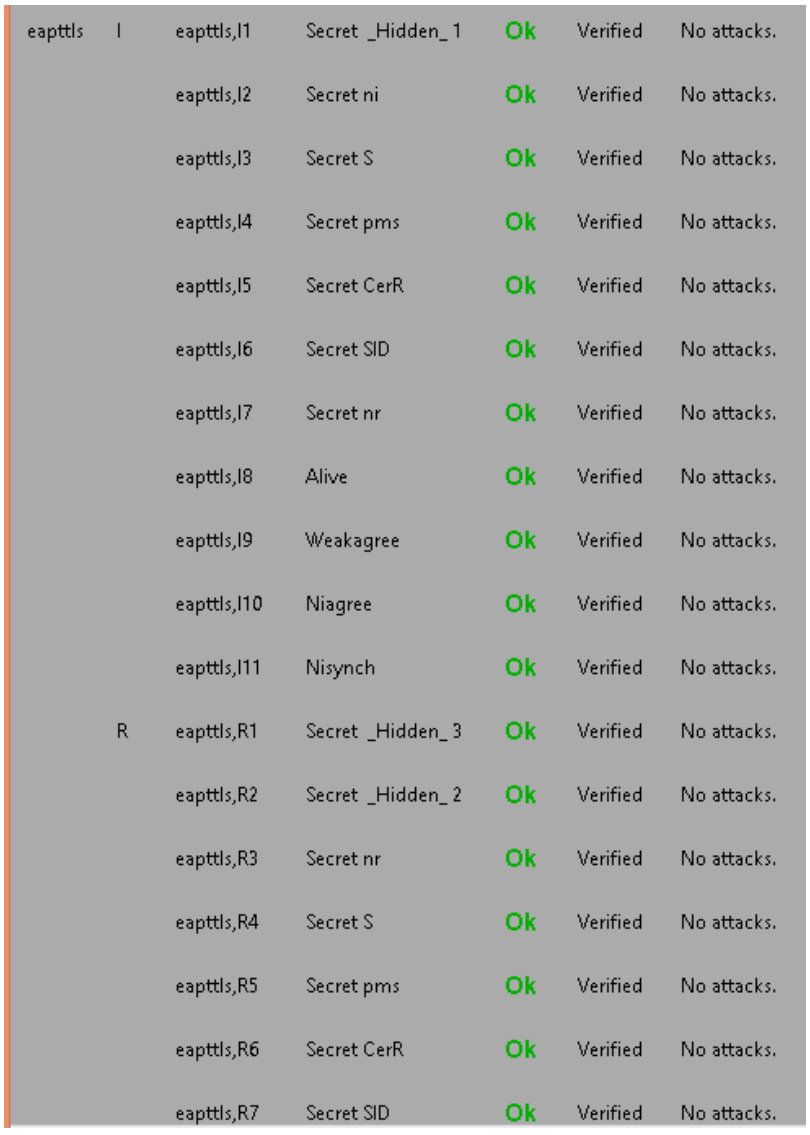
claim_r1 (R, Secret, ni);
claim_r2 (R, Secret, nr);
claim_r3 (R, Secret, CerR);
claim_r4 (R, Secret, ms);

```
claim_r5 (R, Secret, clientkey);
claim_r6 (R, Secret, serverkey);
claim_r7 (R, Secret, chapchallenge);
claim_r8 (R, Secret, password);
claim_r9 (R, Nisynch);
claim_r10 (R, Niagree);
}

role CA
{
const CerR: Data;
recv_2 (R, CA, R);
send_3 (CA, R, {R, {CerR, pk(R)}pk(R)}sk(CA));
}}
```



eapttls	I	eapttls,i1	Secret ni	Ok	Verified	No attacks.
		eapttls,i2	Secret nr	Ok	Verified	No attacks.
		eapttls,i3	Secret CerR	Ok	Verified	No attacks.
		eapttls,i4	Secret prf(pms,mastersecret,ni,nr)	Ok	Verified	No attacks.
		eapttls,i5	Secret prf(prf(pms,mastersecret,ni,nr),ttlskeyingm...	Ok	Verified	No attacks.
		eapttls,i6	Secret prf(prf(pms,mastersecret,ni,nr),ttlskeyingm...	Ok	Verified	No attacks.
		eapttls,i7	Secret h(prf(pms,mastersecret,ni,nr),ttlschallenge...	Ok	Verified	No attacks.
		eapttls,i8	Secret password	Ok	Verified	No attacks.
		eapttls,i9	Nisynch	Ok	Verified	No attacks.
		eapttls,i10	Niagree	Ok	Verified	No attacks.
	R	eapttls,r1	Secret ni	Ok	Verified	No attacks.
		eapttls,r2	Secret nr	Ok	Verified	No attacks.
		eapttls,r3	Secret CerR	Ok	Verified	No attacks.
		eapttls,r4	Secret prf(pms,mastersecret,ni,nr)	Ok	Verified	No attacks.
		eapttls,r5	Secret prf(prf(pms,mastersecret,ni,nr),ttlskeyingm...	Ok	Verified	No attacks.
		eapttls,r6	Secret prf(prf(pms,mastersecret,ni,nr),ttlskeyingm...	Ok	Verified	No attacks.
		eapttls,r7	Secret h(prf(pms,mastersecret,ni,nr),ttlschallenge...	Ok	Verified	No attacks.
		eapttls,r8	Secret password	Ok	Verified	No attacks.

Fig. 9
"Scyther analysis of EAP-TTLS for claims specified by us"



eapttls	I	eapttls,I1	Secret _Hidden_1	Ok	Verified	No attacks.
		eapttls,I2	Secret ni	Ok	Verified	No attacks.
		eapttls,I3	Secret S	Ok	Verified	No attacks.
		eapttls,I4	Secret pms	Ok	Verified	No attacks.
		eapttls,I5	Secret CerR	Ok	Verified	No attacks.
		eapttls,I6	Secret SID	Ok	Verified	No attacks.
		eapttls,I7	Secret nr	Ok	Verified	No attacks.
		eapttls,I8	Alive	Ok	Verified	No attacks.
		eapttls,I9	Weakagree	Ok	Verified	No attacks.
		eapttls,I10	Niagree	Ok	Verified	No attacks.
		eapttls,I11	Nisynch	Ok	Verified	No attacks.
	R	eapttls,R1	Secret _Hidden_3	Ok	Verified	No attacks.
		eapttls,R2	Secret _Hidden_2	Ok	Verified	No attacks.
		eapttls,R3	Secret nr	Ok	Verified	No attacks.
		eapttls,R4	Secret S	Ok	Verified	No attacks.
		eapttls,R5	Secret pms	Ok	Verified	No attacks.
		eapttls,R6	Secret CerR	Ok	Verified	No attacks.
		eapttls,R7	Secret SID	Ok	Verified	No attacks.

Fig. 10
"Scyther analysis of EAP-TTLS through Autoverify"

3.4.3 Timing Analysis in MATLAB

trsae= 0.1667 %rsaencryption
trsad= 0.1667 %rsadecryption
thmacsha256= 0.0714

tsha1= 0.0384; %sha
tsha256= 0.05 %sha256
tdese= 0.29165; %desencryption
tdesd= 0.29165; %desdecryption
tmd4= 0.0476

Original EAP-TTLS

- trsae
- trsad
- 5*thmacsha256+2*thmacsha256
- 5*thmacsha256+2*thmacsha256
- 2*thmacsha256
- 2*thmacsha256
- 14*thmacsha256+2*thmacsha256+tdese
- 14*thmacsha256+2*thmacsha256+tdesd
- tdese
- tdesd

Modified EAP-TTLS

- trsae
- trsad
- 2*thmacsha256+tsha1+thmacsha256
- 2*thmacsha256+tsha1+thmacsha256
- thmacsha256
- thmacsha256
- 4*thmacsha256+6*tsha1+tmd4+tdese
- 4*thmacsha256+tsha1+tmd4+tdesd
- tdese
- tdesd

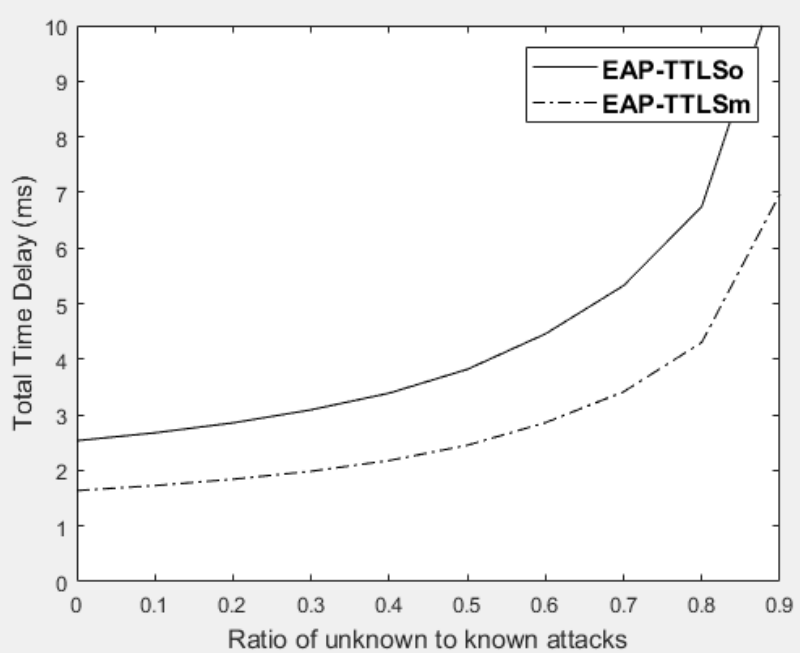


Fig. 11

3.5 PEAP (Protected Extensible Authentication Protocol)

PEAP does not specify an authentication method, but provides additional security for other EAP authentication protocols, such as EAP-MSCHAP v2, that can operate through the TLS encrypted channel provided by PEAP. PEAP protocol works as follows:

- A TLS channel that provides protection for the EAP method negotiation that occurs between client and server. This TLS channel helps prevent an attacker from injecting packets between the client and the network access server to cause the negotiation of a less secure EAP type. The key that is derived during this negotiation is used to encrypt all subsequent communication, including network access authentication that allows the user to connect to the organization network.
- Clients with the ability to authenticate the RADIUS server. Because the server also authenticates the client, mutual authentication occurs.
- PEAP fast reconnect reduces the delay between an authentication request by a client and the response by the RADIUS server. Fast reconnect also allows wireless clients to move between access points that are configured as RADIUS clients to the same RADIUS server without repeated requests for authentication. This reduces resource requirements for both client and server, and minimizes the number of times that users are prompted for credentials.

3.5.1 PEAP analysis through Scyther

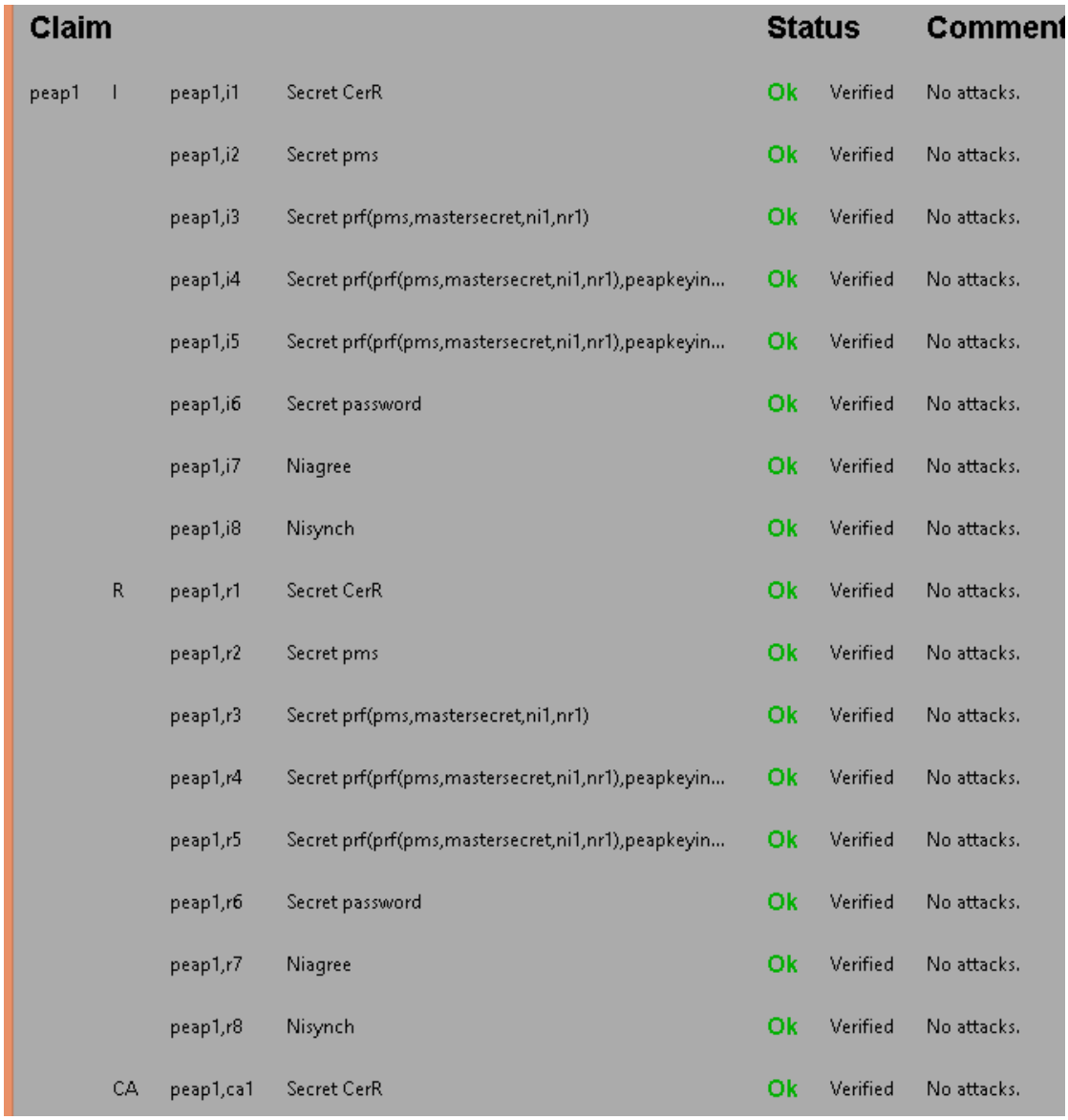
```
hashfunction prf;
hashfunction h1;
hashfunction h;

usertype String;
const ccs, csu, shd, mastersecret, password : String;
const PEAPsuccess, accesslevel, clientfinished, serverfinished,
peapkeyingmaterial,uiname,urname : String;

macro ms= prf (pms, mastersecret, (ni1, nr1));
macro finishedi= h1 (ms, clientfinished, (SID, ni1, nr1, csu, {CerR, pk(R)}sk(CA), shd, {pms}pk(R)));
```

```
macro finishedr= h1(ms, serverfinished, (SID, ni1, nr1, csu, {CerR, pk(R)}sk(CA), shd, {pms}pk(R), finishedi));
macro ikey= prf ( ms, peapkeyingmaterial, (ni1, nr1));
macro skey= prf ( ms, peapkeyingmaterial, (ni1, nr1));
macro hash= h(n, password);

protocol  peap1(I, R, CA)
{
role I
{
const SID, ID, CerR, pms: Data;
fresh ni1: Nonce;
var nr1, n: Nonce;

send_1 (I, R, SID, ni1, csu);
recv_4 (R, I, SID, nr1, csu, {CerR, pk(R)}sk(CA), shd);
send_5 (I, R, {pms}pk(R), ccs);
send_6 (I,R, finishedi);
recv_7 (R, I, ccs,finishedr);
match ( h1(ms, serverfinished, (SID, ni1, nr1, csu, {CerR, pk(R)}sk(CA), shd, {pms}pk(R), finishedi)), finishedr);
send_8 (I,R, {uiname}ikey);
recv_9 (R, I, {ID, n, uiname}skey);
send_10 (I, R, {ID, h(n, password), urname}ikey);
recv_11 (R, I, {PEAPsuccess, accesslevel}skey);

claim_i1 (I, Secret, CerR);
claim_i2 (I, Secret, pms);
claim_i3 (I, Secret, ms);
claim_i4 (I, Secret, ikey);
claim_i5 (I, Secret, skey);
claim_i6 (I, Secret, password);
claim_i7 (I, Niagree);
claim_i8 (I, Nisynch);
}
role R
{
```

```
const  SID, ID, CerR, pms: Data;
fresh nr1, n: Nonce;
var ni1: Nonce;

recv_1 (I, R, SID, ni1, csu);
send_2 (R, CA, R);
recv_3 (CA, R, {R, {CerR, pk(R)}pk(R)} sk(CA));
send_4 (R, I, SID, nr1, csu, {CerR, pk(R)}sk(CA), shd);
recv_5 (I, R, {pms}pk(R), ccs);
recv_6 (I,R, finishedi);
match (h1 (ms, clientfinished, (SID, ni1, nr1, csu, {CerR, pk(R)}sk(CA), shd, {pms}pk(R))),
finishedi);
send_7 (R, I, ccs, finishedr);
recv_8 (I,R, {uiname}ikey);
send_9 (R, I, {ID, n, uiname}skey);
recv_10 (I, R, {ID, h(n, password), urname}ikey);
match (hash, h(n,password));
send_11(R, I, {PEAPsuccess, accesslevel}skey);

claim_r1 (R, Secret, CerR);
claim_r2 (R, Secret, pms);
claim_r3 (R, Secret, ms);
claim_r4 (R, Secret, ikey);
claim_r5 (R, Secret, skey);
claim_r6 (R, Secret, password);
claim_r7 (R, Niagree);
claim_r8 (R, Nisynch);
}
role CA
{
const CerR: Data;
recv_2(R, CA, R);
send_3 (CA, R, {R, {CerR, pk(R)}pk(R)} sk(CA));
claim_ca1 (CA, Secret, CerR);
} }
```

peap1	I	peap1,I1	Secret _Hidden_ 1	Ok	Verified	No attacks.
		peap1,I2	Secret ni1	Ok	Verified	No attacks.
		peap1,I3	Secret pms	Ok	Verified	No attacks.
		peap1,I4	Secret CerR	Ok	Verified	No attacks.
		peap1,I5	Secret ID	Ok	Verified	No attacks.
		peap1,I6	Secret SID	Ok	Verified	No attacks.
		peap1,I7	Secret n	Ok	Verified	No attacks.
		peap1,I8	Secret nr1	Ok	Verified	No attacks.
		peap1,I9	Alive	Ok	Verified	No attacks.
		peap1,I10	Weakagree	Ok	Verified	No attacks.
		peap1,I11	Niagree	Ok	Verified	No attacks.
		peap1,I12	Nisynch	Ok	Verified	No attacks.
	R	peap1,R1	Secret _Hidden_ 3	Ok	Verified	No attacks.
		peap1,R2	Secret _Hidden_ 2	Ok	Verified	No attacks.
		peap1,R3	Secret n	Ok	Verified	No attacks.
		peap1,R4	Secret nr1	Ok	Verified	No attacks.
		peap1,R5	Secret pms	Ok	Verified	No attacks.
		peap1,R6	Secret CerR	Ok	Verified	No attacks.

Fig. 12
"Scyther analysis of PEAP with claims specified by us"



Claim				Status		Comment
peap1	I	peap1,i1	Secret CerR	Ok	Verified	No attacks.
		peap1,i2	Secret pms	Ok	Verified	No attacks.
		peap1,i3	Secret prf(pms,mastersecret,ni1,nr1)	Ok	Verified	No attacks.
		peap1,i4	Secret prf(prf(pms,mastersecret,ni1,nr1),peapkeyin...	Ok	Verified	No attacks.
		peap1,i5	Secret prf(prf(pms,mastersecret,ni1,nr1),peapkeyin...	Ok	Verified	No attacks.
		peap1,i6	Secret password	Ok	Verified	No attacks.
		peap1,i7	Niagree	Ok	Verified	No attacks.
		peap1,i8	Nisynch	Ok	Verified	No attacks.
	R	peap1,r1	Secret CerR	Ok	Verified	No attacks.
		peap1,r2	Secret pms	Ok	Verified	No attacks.
		peap1,r3	Secret prf(pms,mastersecret,ni1,nr1)	Ok	Verified	No attacks.
		peap1,r4	Secret prf(prf(pms,mastersecret,ni1,nr1),peapkeyin...	Ok	Verified	No attacks.
		peap1,r5	Secret prf(prf(pms,mastersecret,ni1,nr1),peapkeyin...	Ok	Verified	No attacks.
		peap1,r6	Secret password	Ok	Verified	No attacks.
		peap1,r7	Niagree	Ok	Verified	No attacks.
		peap1,r8	Nisynch	Ok	Verified	No attacks.
	CA	peap1,ca1	Secret CerR	Ok	Verified	No attacks.

Fig. 13
"Scyther Analysis of PEAP through Autoverify"

3.5.2 Timing Analysis in MATLAB

```
trsae= 0.1667 %rsaencryption
trsad= 0.1667 %rsadecryption
thmacsha256= 0.0714
tsha1= 0.0384; %sha
tsha256= 0.05 %sha256
tmd5= 0.05263; %md5
```

tdese= 0.29165;
tdesd= 0.29165;

Original PEAP

- trsae
- trsad
- 5*thmacsha256+2*thmacsha256
- 5*thmacsha256+ 2*thmacsha256
- 2*thmacsha256
- 2*thmacsha256
- 14*thmacsha256 + tdese
- 14*thmacsha256+tdesd
- tdese
- tdesd
- tmd5+tdese
- tdesd+tmd5
- tdese
- tdesd

Modified PEAP

- trsae
- trsad
- 2*thmacsha256+tsha1+thmacsha256
- 2*thmacsha256+tsha1+thmacsha256
- thmacsha256
- thmacsha256
- 4*thmacsha256+6*tsha1+tdese
- 4*thmacsha256+6*tsha1+tdesd
- tdese
- tdesd
- tmd5+tdese
- tmd5+tdese
- tdese
- tdesd

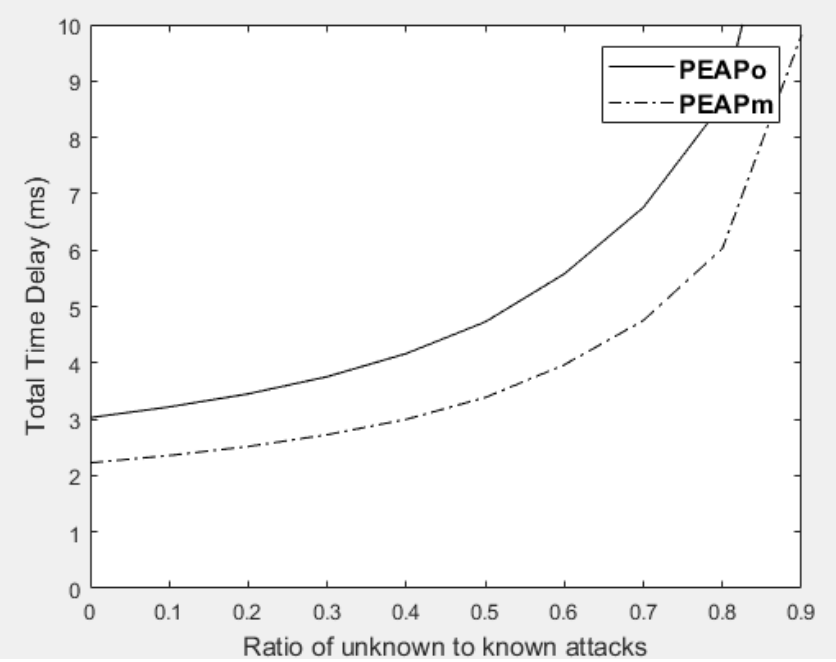


Fig.14
"MATLAB analysis on efficiency of PEAP"

CHAPTER 4
RESULT ANALYSIS

In the previous chapter, methodology, I have shown the preliminary results obtained on analysis through Scyther and MATLAB, the analysis of which will be presented in this chapter.

4.1 MS-CHAP V2 with LDAP Result Analysis

Firstly, since we identified MSCHAP v2 with LDAP to be a weak protocol, we proved in Scyther its vulnerabilities and we can see that there are more than a 100 ways in which the protocol can be weakened. We could clearly see that the root cause of most of the vulnerabilities is due to the sending of username and nonces (challenge-response mechanism) in clear text which makes it vulnerable to ARP-injection attacks (In ARP spoofing, the attacker associates his MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead. It also allows attackers to intercept data frames on a network, modify the traffic, or stop all traffic. Often, the attack is an entering point to employ man-in-the-middle and DoS attacks). Moreover, complex tools such as ASLEAP have been designed to recover weak MS-CHAP passwords through dictionary attacks (brute force techniques) using the same methods of MD-4 and DES encryption.

Hence, in our revision of the protocol we encrypt the username and nonces through the use of asymmetric keys. Further, the original MS-CHAP only employs MD-4 and SHA-1 for hashing, both have been proved weak i.e. are not resistant to pre-image attacks (through which given a hash, we can find a message that has that hash) and are also vulnerable to collision attacks (multiple messages having the same hash value), therefore we use SHA-256 in the computation of challenge response .

However, when we make changes to improve security of the protocol, we lose out on efficiency and the time taken by the protocol under multiple known and unknown attacks rises from approx. 1.5ms to 2.8 ms.

Since this does not seem like a feasible option, we explore more algorithms to be employed in AuthFlow which can not only provide more security, but also take lesser time for execution.

4.2 EAP-TLS Result analysis

If administrative/ economic constraints are removed, we find that EAP-TLS seems the perfect choice for AuthFlow. (If there is availability of X.509 certificates which meet the requirements of the Internet Authentication Service, CryptoAPI certificate store and can pass the checks for Secure Sockets Layer encryption and Transport Layer Security encryption). We confirm with Scyther, that EAP-TLS is secure in terms of protocol design and is not vulnerable to any attacks. The only disadvantage with EAP-TLS was the time consumed which was approx. 2.6ms, hence we work to improve efficiency/reduce time taken by the execution of the protocol.

Changes made to the EAP-TLS protocol for execution in the AuthFlow infrastructure:

The max. time taken in the protocol is in the hash calculation through Pseudo Random functions. Moreover, the elaborate hashing mechanism is used after the exchange of keying parameters and are only used to cross-check the computed parameters on each side before exchange of information.

PRF(secret, label, seed) = P_<hash>(secret, label + seed) where P_hash is:

P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) + HMAC_hash(secret, A(2) + seed) +
HMAC_hash(secret, A(3) + seed) + ... // where + indicates concatenation

And A() is defined as:

A(0) = seed

A(i) = HMAC_hash(secret, A(i-1))

Older versions of EAP-TLS used MD5+SHA1 for calculating A(i), however since we are using HMAC-SHA256 which is resistant to pre-image and collision attacks, we can use SHA1 in A() calculations, as layered encryption is employed in PRF.

Here, a significant difference is also obtained by starting with A(0) itself and then incrementing. (reduces one hashing in each step, and since our algorithm requires 128 or 160 bits each time, security is not an issue)

P_hash(secret, seed) = HMAC_SHA256(secret, A(0) + seed) + HMAC_ SHA256 (secret, A(1) + seed) + HMAC_ SHA256 secret, A(2) + seed) + ...

A() is defined as:

A(0) = seed
A(i) = SHA1(secret, A(i-1))

key_block = PRF(SecurityParameters.master_secret, "key expansion",
SecurityParameters.server_random + SecurityParameters.client_random);

Key_block is repeated until the reuired output i.e. no. of bits have been generated.

And it is partitioned as follows:

client_write_MAC_key[SecurityParameters.mac_key_length]
server_write_MAC_key[SecurityParameters.mac_key_length]
client_write_key[SecurityParameters.enc_key_length]
server_write_key[SecurityParameters.enc_key_length]
client_write_IV[SecurityParameters.fixed_iv_length]
server_write_IV[SecurityParameters.fixed_iv_length]

After the implementation of these changes we can observe that the time is reduced to 1.8 ms and the security of the protocol can also be mathematically proved through BAN logic.

However, in the absence of client/server side certificate, we can alternatively use EAP-TTLS and PEAP mechanisms.

4.3 EAP-TTLS Result analysis

In the EAP-TTLS mechanism considered we secure the server side through TLS and after exchange of keying material, the client side is authenticated through CHAP. The security of the proposed algorithm is verified through Scyther and is secure form any attacks. However, the efficiency of the protocol is improved as the current execution takes 2.5ms. Hence, we change the PRF function as described in EAP-TLS for the TLS version of the protocol and then use MD-4 hashing rather than complex hashing methods as it also encrypted under the pre-calculated client key computed with the parameters exchanged during TLS reducing the time taken by the protocol to 1.7ms.

4.4 PEAP Result analysis

In certain scenarios where we are working only with Microsoft networks and would be using only EAP methods for authentication, PEAP can be used which works on a mechanism similar to EAP-TTLS where keying material/security parameters are exchanged in the first phase and authentication is performed in the second stage. This protocol's security has been verified through Scyther. However, execution in MATLAB shows that the protocol takes an average of 3ms under known/unknown attacks. On employing methods similar to those used in EAP-TLS we can improve the efficiency of the protocol and reduce time taken by it to 2.2ms without effecting security of the protocol.

In this project, we have therefore identified shortcomings in MS-CHAP v2 with LDAP, made it more secure through better encryption and hashing schemes, however due to the time taken by the improved protocol we explore more protocols which can replace MSCHAP v2 with LDAP in AuthFlow. In the availability of certificates we find, that EAP-TLS, EAP-TTLS and PEAP are all secure protocols whose efficiency has been increased through simple changes in the design of hashing algorithms, which can be proved in MATLAB on comparing the original and modified protocols under continuous known/unknown attacks. Hence, we have achieved the aim of making AuthFlow and hence SDN more secure and efficient through the implementation of our protocols.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE OF WORK

5.1 Work Conclusion

Our aim of making the open programmable SDN infrastructure more secure and efficient has been achieved through our analysis and implementation of MS-CHAP v2 with LDAP, EAP-TLS, EAP-TTLS and PEAP. The unique work methodology adopted included a combination of mathematical proof through BAN logic, automated analysis through Scyther, time calculation for individual cipher algorithms through C/C++ and analysis of efficiency of security protocols under continuous known/unknown attacks through MATLAB.

We succeeded in making the MS-CHAP v2 protocol more secure but it resulted in loss of efficiency. Hence, future scope includes working on the secure version of MS-CHAP v2, changing some encryption/hashing techniques so as to reduce time taken for execution of protocol because it is the only mechanism currently in use for AuthFlow which does not require the exchange of certificates as access credentials.

If there was no administrator/economic constraint for the security protocol employed, modified EAP-TLS would be the most secure and efficient mechanism used in AuthFlow. The security of which has been proved in BAN, verified through Scyther, and efficiency improved and verified through MATLAB analysis.

EAP-TTLS and PEAP have been proved to be both secure and efficient in our revised versions. But research is fundamental, it is about finding as much as you can and never giving up, therefore I believe more work can be done on this front to design algorithms which while being secure can work with implementations which do not require repeated hashing such as in pseudo random functions and multiple encryption layers.

Moreover, through our extensive literature survey, we have identified some ares in SDN security which require improvement and recommendations for the same are described below.

5.2 Recommendations to be considered for SDN deployment today/ Future Scope

- Policy Conflict Resolution/ Network Invariant Detection: When application modules manipulate the network state, the controller should identify misconfigurations, unauthorized access, and irregularities to ensure correct functioning of the entities.

- Mutual Authentication: SDN components should enable authentication solutions both within and across trusted domains to avoid the introduction of insecure access to network resources. This prevents data manipulation attacks, impersonation of components and ensures secure identification of network entities.

- Control Plane Isolation via Slicing: Unlike network virtualization, slicing the network resources will partition the resources allocated for users sharing the infrastructure. From a security standpoint, this isolated environment can be securely instantiated and protected from unauthorized access, data manipulations and data leakages.

- Containerized Applications: Assessing the different controller implementations, network applications are either statically compiled with the controller code, instantiated as a dynamic module with the controller software or integrated as a third party software with remote access to the controller. To restrict the impact of malicious application behaviour, it is recommended to support application containerization, which can authenticate the application during setup, control the application's access rights on the infrastructure (bandwidth, latency, counters to monitor etc.), and limit, account for and isolate the resource usage for each application.

- Rate-Limiting, Flow Aggregation and Short Timeouts: The correct use of flow and switch attributes such as flags, timeouts, modes of operation as well as the inherent security features such as metering to rate-limit the data flow to the control plane can ensure correct packet forwarding behaviour by avoiding overlaps, notifying flow deletes and operating securely when the connection is lost with the controller.

- Logging/ Forensics for IDS/IPS*:* Network services and applications with monitoring capabilities require logging critical information which helps in troubleshooting and debugging the infrastructure. They should be able to determine network state at any given time and be able to track back to the network state at a previous point in time.

References

1. S. Scott-Hayward, S. Natarajan, and S. Sezer, “A survey of security in software defined networks,” IEEE Communications Surveys & Tutorials, 2016.
2. Diogo Menezes Ferrazani Mattos, Otto Carlos Muniz Bandeira Duarte, “AuthFlow: Authentication and access control mechanism for Software Defined Networking”, Article in anals of Communication, 2016.
3. Michael Burrows, Martin Abadi and Roger Needham, “A logic of authentication”, ACM Transactions on Computer Systems, Vol. 8, No. 1, 1990.
4. Cas Cremers and Sjouke Mauw. “Operational Semantics and Verification of Security Protocols”, Information Security and Cryptography, 2012.
5. https://tools.ietf.org/html/rfc2759 (MS-CHAP v2)
6. https://tools.ietf.org/html/rfc5216 (EAP-TLS)
7. https://tools.ietf.org/html/rfc5246 (TLS)
8. https://tools.ietf.org/html/rfc5281 (EAP-TTLS)
9. https://tools.ietf.org/id/draft-josefsson-pppext-eap-tls-eap-06.txt (PEAP)
10. Noudjoud Kahya, Nacira Ghoulami, Pascal Lafourcade, “Formal Analysis of PKM using Scyther Tool”, International Conference on Information Technology and e-Services, 2012.
11. Seong-Soo Park, Jong-Hyouk Lee, and Tai-Myoung Chung, “Authentication Analysis Based on Certificate for Proxy Mobile IPv6 Environment”, ICCSA '09 Proceedings of the International Conference on Computational Science and Its Applications, Part I, 2009.
12. Yue Qiu and Maode Ma, “An Efficient and Secure PMIPv6-BASED Handover Scheme for 6LoWPAN Networks”. (In Review).
13. https://technet.microsoft.com/en-us/library
14. https://www.programmersheaven.com
15. https://github.com/
16. http://rosettacode.org/wiki/Rosetta_Code
17. http://www.zedwood.com/ (C++ hashing)

PROJECT DETAILS

Student Details			
Student Name	**Garuda Suma Pranavi**		
Register Number	130907120	Section / Roll No	A/13
Email Address	suma.garuda@yahoo.com	Phone No (M)	9899208440
Project Details			
Project Title	**Investigation of Network Topology Poisoning Attacks in SDN**		
Project Duration	5 months	Date of reporting	26th Jan 2017
Expected date of completion of project	24th June 2017		
Organization Details			
Organization Name	**Nanyang Technological University, Singapore**		
Full postal address with pin code	**Nanyang Technological University,** **50 Nanyang Avenue, Singapore- 639798**		
Website address	**http://www.ntu.edu.sg/Pages/home.aspx**		
Supervisor Details			
Supervisor Name	**Prof. Ma Maode**		
Designation	**Associate Professor**		
Full contact address with pin code	**S2. 2-B2-27, School of Electrical and Electronic Engineering Nanyang Technological University, 50 Nanyang Avenue, Singapore**		
Email address	**EMDMa@ntu.edu.sg**	Phone No (M)	**+65 67904385**
Internal Guide Details			
Faculty Name	**Prof. Stanley Oswald Maben**		
Full contact address with pin code	**Dept. of E&C Engg., Manipal Institute of Technology, Manipal – 576 104, Karnataka, INDIA**		
Email address	**stan.maben@manipal.edu**		