Security analysis of EAP-TLS and EAP-FAST using Scyther Tool

A Project Report

Submitted by:

Shubham Kumar Verma

Enrollment Number -21535028

MASTER OF TECHONOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

at

IIT ROORKEE

November 7, 2021

DECLARATION

I hereby declare that the project entitled "**Security analysis of EAP-TLS and EAP-FAST using Scyther Tool**"submitted for the M. Tech. Project Lab is my original work.

Signature of the Student

Place: Roorkee

Date:07/11/2021

CERTIFICATE

This is to certify that the project titled "**Security analysis of EAP-TLS and EAP-FAST using Scyther Tool**"is the bona fide work carried out by **SHUBHAM KUMAR VERMA**, a student of **M Tech (CSE) 1st year of IIT Roorkee** during the academic year 2021-22.

Signature of the Guide

Place: Roorkee
Date: 07/11/ 2021

Abstract

With this project Security analysis of EAP-TLS and EAP-FAST using Scyther Tool, I tried to do security analysis of EAP-TLS and EAP-FAST authentication protocols.

Introduction

Problem Definition

In this project I tried to do security analysis of authentication protocols EAP-TLS and EAP-FAST using SCYTHER tool. Below is the introduction of each of them.

SCYTHER

Scyther Tool

Scyther is a tool for the automatic verification of security protocols. Some interesting features are:

1. Scyther can verify protocols with an unbounded number of sessions and nonce
2. Scyther can characterize protocols, yielding a finite representation of all possible protocol behaviours.
3. It is efficient.

Protocols analysed with Scyther

1. Scyther has been used to analyse the IKEv1 and IKEv2 protocol suites and the ISO/IEC 9798 family of authentication protocols.
2. The tool has also been used to find new multi-protocol attacks on many existing protocols.

3. The tool comes bundled with a set of example protocol files modelled from the SPORE repository, which is set up to be an online continuation of the Clark-Jacob library of authentication protocols.

Installation Steps:

Scyther uses some other components, specified below.

1. The GraphViz library:-

   This library is used by the Scyther tool to draw graphs. It can be downloaded from:

   http://www.graphviz.org/download/

2. Python

   The graphical user interface of Scyther is written in the Python language.Python 3 not supported

   http://www.python.org/download/

3. wxPython libraries

   The graphical user interface of Scyther using the wxPython library to draw widgets.

   http://www.wxpython.org/download.php

Running Scyther

Start Scyther by executing the file: scyther-gui.py

EAP-TLS

Introduction

Extensible Authentication Protocol – Transport Layer Security (EAP-TLS) is an IETF open standard that's defined in RFC 5216. EAP-TLS is the authentication protocol most commonly deployed on WPA2 Despite being the pinnacle of authentication security, EAP-TLS remains a relatively simple framework for authentication. It doesn't rely on overly complicated encryption schemes or anything like that – it's predicated on the strength of public key cryptography. Enterprise networks to enable the use of X.509 digital certificates for authentication.

Terminology (taken from RFC-5216)

authenticator
The entity initiating EAP authentication.

peer
The entity that responds to the authenticator. In [IEEE-802.1X], this entity is known as the Supplicant.

backend authentication server
A backend authentication server is an entity that provides an authentication service to an authenticator. When used, this server typically executes EAP methods for the authenticator. This terminology is also used in [IEEE-802.1X].

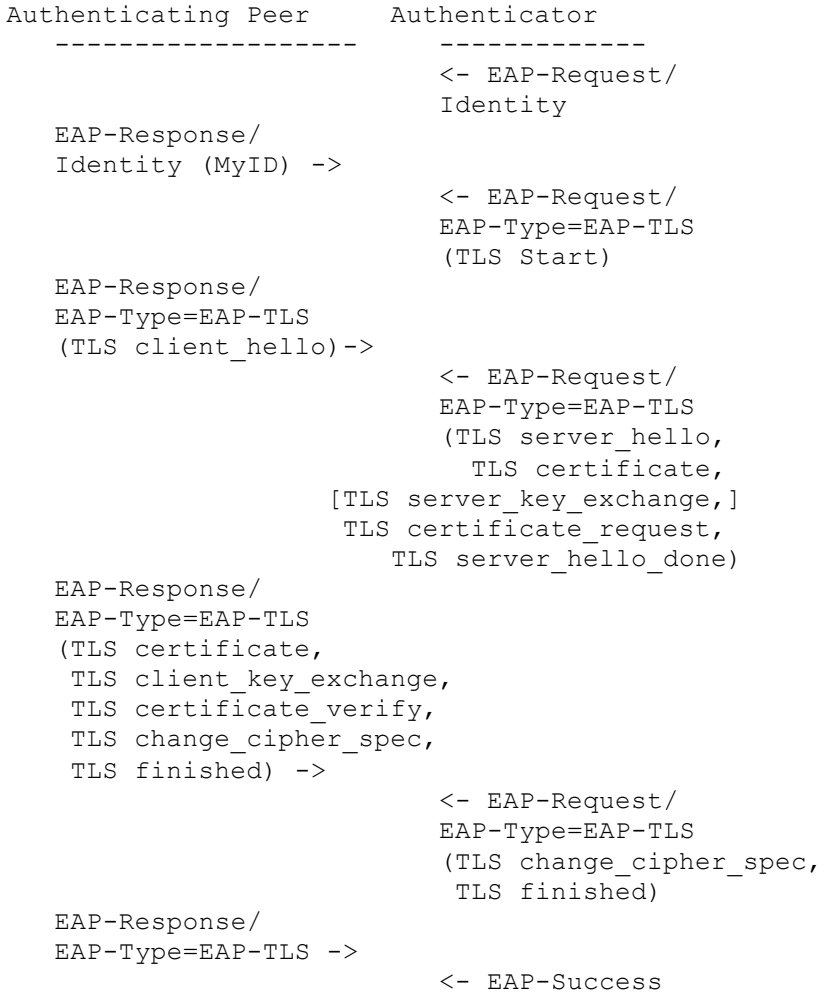
EAP server
The entity that terminates the EAP authentication method with the peer. In the case where no backend authentication server is used, the EAP server is part of the authenticator. In the case were the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.

Master Session Key (MSK)
Keying material that is derived between the EAP peer and server and exported by the EAP method.

Extended Master Session Key (EMSK)
Additional keying material derived between the EAP peer and server that is exported by the EAP method.

Working of EAP-TLS

The EAP-TLS conversation will typically begin with the authenticator and the peer negotiating EAP. The authenticator will then typically send an EAP-Request/Identity packet to the peer, and the peer will respond with an EAP-Response/Identity packet to the authenticator, containing the peer's user-Id. From this point forward, while nominally the EAP conversation occurs between the EAP authenticator and the peer, the authenticator MAY act as a pass-through device, with the EAP packets received from the peer being encapsulated for transmission to a backend authentication server. This is working of EAP-TLS



EAP-FAST

Introduction

EAP-FAST is an EAP method that enables secure communication between a peer and a server by using the Transport Layer Security (TLS) to establish a mutually authenticated tunnel. Within the tunnel, Type-Length-Value (TLV) objects are used to convey authentication related data between the peer and the EAP server.

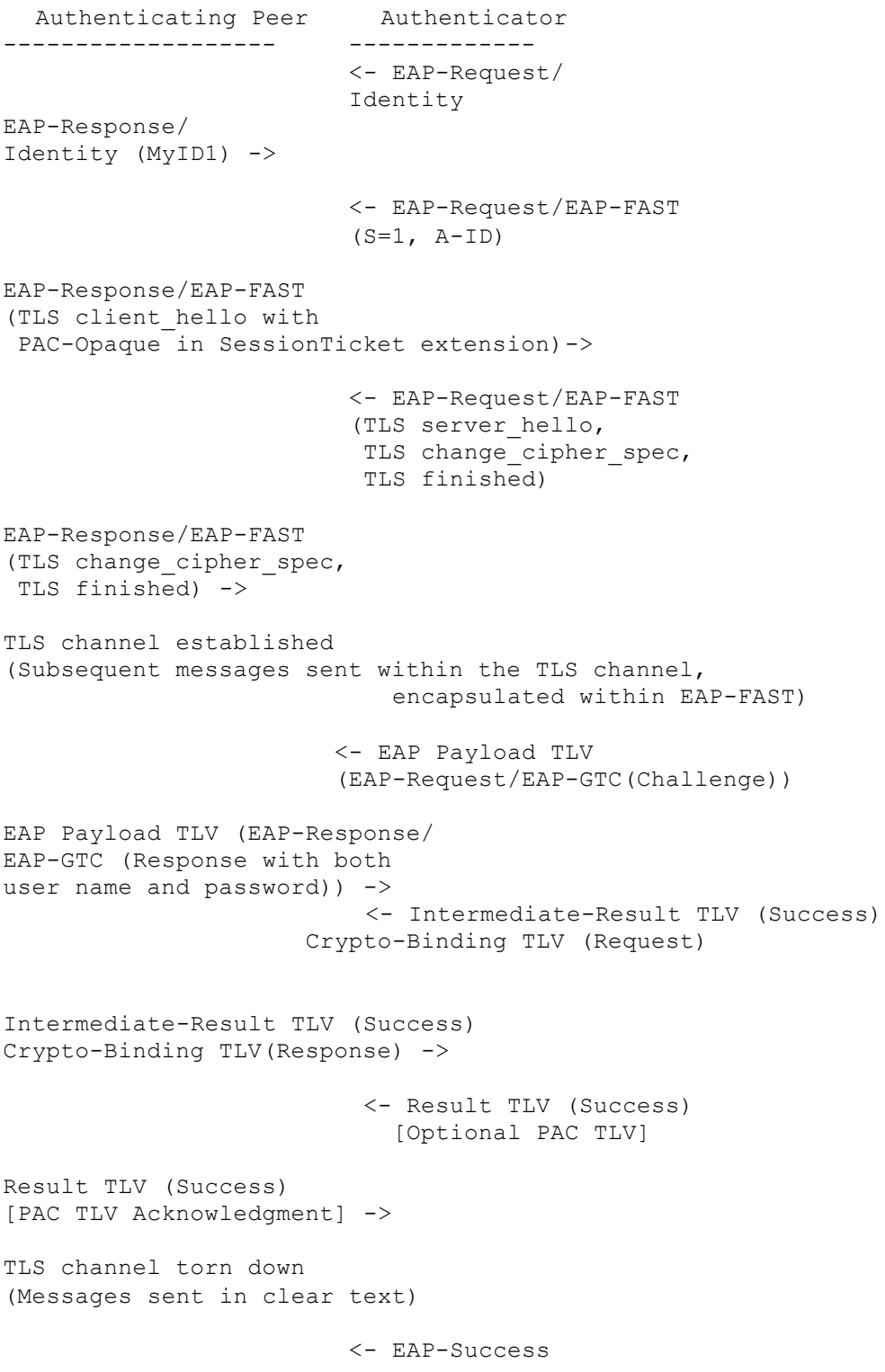
Protocol Overview.

EAP-FAST is an authentication protocol similar to EAP-TLS [RFC2716] that enables mutual authentication and cryptographic context establishment by using the TLS handshake protocol. EAP-FAST allow for the established TLS tunnel to be used for further authentication exchanges. EAP-FAST makes use of TLVs to carry out the inner authentication exchanges. The tunnel is then used to protect weaker inner authentication methods, which may be based on passwords, and to communicate the results of the authentication.

EAP-FAST makes use of the TLS enhancements in [RFC4507] to enable an optimized TLS tunnel session resume while minimizing server state.The secret key used in EAP-FAST is referred to as the Protected Access Credential key (or PAC-Key); the PAC-Key is used to mutually authenticate the peer and the server when securing a tunnel. The ticket is referred to as the Protected Access Credential opaque data (or PAC-Opaque). The secret key and ticket used to establish the tunnel may be provisioned through mechanisms that do not involve the TLS handshake. It is RECOMMENDED that implementations support the capability to distribute the ticket and secret key within the EAP- FAST tunnel as specified in [EAP-PROV].

The EAP-FAST conversation is used to establish or resume an existing session to typically establish network connectivity between a peer and the network. Upon successful execution of EAP-FAST, both EAP peer and EAP server derive strong session key material that can then be communicated to the network access server (NAS) for use in establishing a link layer security association.

Working


Authenticating Peer
Authenticator
<- EAP-Request/
Identity
EAP-Response/
Identity (MyID1) ->
<- EAP-Request/EAP-FAST
(S=1, A-ID)
EAP-Response/EAP-FAST
(TLS client_hello with
PAC-Opaque in SessionTicket extension)->
<- EAP-Request/EAP-FAST
(TLS server_hello,
TLS change_cipher_spec,
TLS finished)
EAP-Response/EAP-FAST
(TLS change_cipher_spec,
TLS finished) ->
TLS channel established
(Subsequent messages sent within the TLS channel,
encapsulated within EAP-FAST)
<- EAP Payload TLV
(EAP-Request/EAP-GTC(Challenge))
EAP Payload TLV (EAP-Response/
EAP-GTC (Response with both
user name and password)) ->
<- Intermediate-Result TLV (Success)
Crypto-Binding TLV (Request)
Intermediate-Result TLV (Success)
Crypto-Binding TLV(Response) ->
<- Result TLV (Success)
[Optional PAC TLV]
Result TLV (Success)
[PAC TLV Acknowledgment] ->
TLS channel torn down
(Messages sent in clear text)
<- EAP-Success

Security Analysis

EAP-TLS

1. Run Scyther-gui.py file using python to open Scyther gui.
2. Click open and select the location where the file eap-tls.spdl is present and click enter.
3. Alternatively, we can write the file in the editor provided
4. Click Verify Protocol.

EAP-TLS.spdl

usertype String;

```
usertype SessionKey;
protocol EAP-TLS(UE,NW)
{

        role UE
        {
        fresh Rue: Nonce;
        fresh Ruel: Nonce;
    fresh Kseaf : SessionKey;
    fresh Methods-UE : String;
    fresh Certificate-UE : String;
         fresh H-HandShake-UE : String;

    var Rnw: Nonce;
    var TLS-Start :String;
    var Methods-NW: String;
    var Certificate-NW:String;
    var H-HandShake-NW:String;
    send_1(UE,NW,{k(UE,NW),Rue}pk(NW));
    recv_2(NW,UE,TLS-Start);
    send_3(UE,NW,Ruel,Methods-UE);
    recv_4(NW,UE,Rnw,{Certificate-NW}sk(NW),Methods-NW);
    send_5(UE,NW,{Kseaf}pk(NW),{Certificate-UE}sk(UE));
    recv_6(NW,UE,{H-HandShake-NW}Kseaf);
    claim_i1(UE,Secret,Kseaf);
    claim_i2(UE,Secret,k(UE,NW));
    claim_i3(UE,Secret,Niagree);
    claim_i4(UE,Secret,Nisynch);
  }

  role NW
  {
    fresh Rnw: Nonce;
    fresh TLS-Start :String;
    fresh Methods-NW: String;
```

```
        fresh Certificate-NW:String;
    fresh H-HandShake-NW:String;
    var Rue: Nonce;
        var Ruel: Nonce;
    var Kseaf : SessionKey;
    var Methods-UE : String;
    var Certificate-UE : String;
    var H-HandShake-UE : String;
    recv_1(UE,NW,{k(UE,NW),Rue}pk(NW));
    send_2(NW,UE,TLS-Start);
    recv_3(UE,NW,Ruel,Methods-UE);
    send_4(NW,UE,Rnw,{Certificate-NW}sk(NW),Methods-NW);
    recv_5(UE,NW,{Kseaf}pk(NW),{Certificate-UE}sk(UE));
    send_6(NW,UE,{H-HandShake-NW}Kseaf);
    claim_i1(NW,Secret,Kseaf);
    claim_i2(NW,Secret,k(UE,NW));
    claim_i3(NW,Secret,Niagree);
    claim_i4(NW,Secret,Nisynch);
  }
}
```

EAP-FAST

1. Run Scyther-gui.py file using python to open scyther gui.
2. Click open and select the location where the file eap-fast.spdl is present and click enter.
3. Alternatively, we can write the file in the editor provided
4. Click Verify Protocol.

EAP-fast.spdl

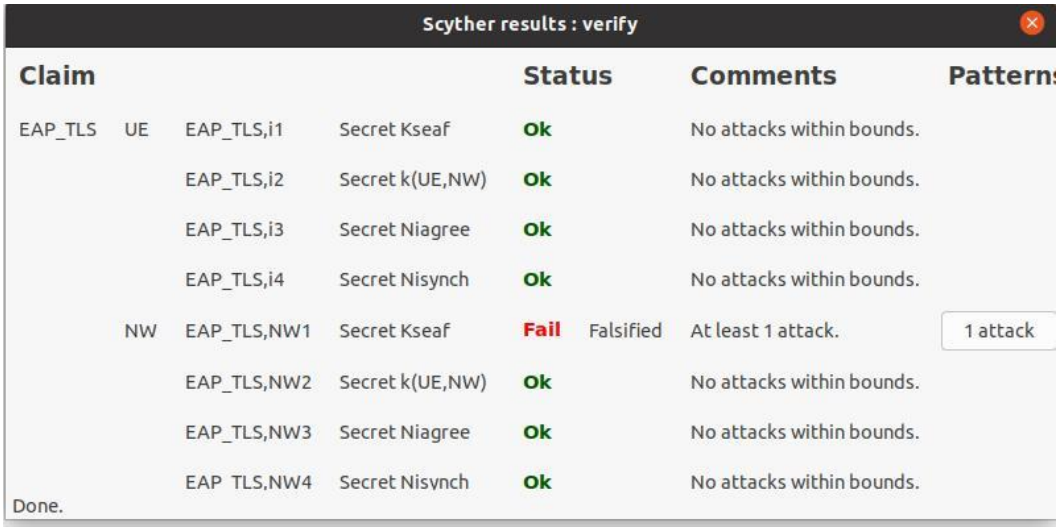
```
usertype String;
usertype SessionKey;
protocol EAP-FAST(UE,NW)
{

        role UE
        {
            fresh Rue: Nonce;
```

```
        fresh Ruel: Nonce;
        fresh Kseaf : SessionKey;
        fresh Methods-UE : String;
        fresh H-HandShake-UE : String;
        var Rnw: Nonce;
        var TLS-Start :String;
        var Methods-NW: String;
        var H-HandShake-NW:String;
        send_1(UE,NW,{k(UE,NW),Rue}pk(NW));
        recv_2(NW,UE,TLS-Start);
        send_3(UE,NW,Ruel,Methods-UE);
        recv_4(NW,UE,{H-HandShake-NW}Kseaf);
        claim_i1(UE,Secret,Kseaf);
        claim_i2(UE,Secret,k(UE,NW));
        claim_i3(UE,Secret,Niagree);
        claim_i4(UE,Secret,Nisynch);
}

role NW
{
   fresh Rnw: Nonce;
   fresh TLS-Start :String;
   fresh Methods-NW: String;
   fresh H-HandShake-NW:String;
   var Rue: Nonce;
   var Ruel: Nonce;
   var Kseaf : SessionKey;
   var Methods-UE : String;
   var H-HandShake-UE : String;
   recv_1(UE,NW,{k(UE,NW),Rue}pk(NW));
   send_2(NW,UE,TLS-Start);
   recv_3(UE,NW,Ruel,Methods-UE);
   send_4(NW,UE,{H-HandShake-NW}Kseaf);
   claim_i1(NW,Secret,Kseaf);
   claim_i2(NW,Secret,k(UE,NW));
   claim_i3(NW,Secret,Niagree);
```

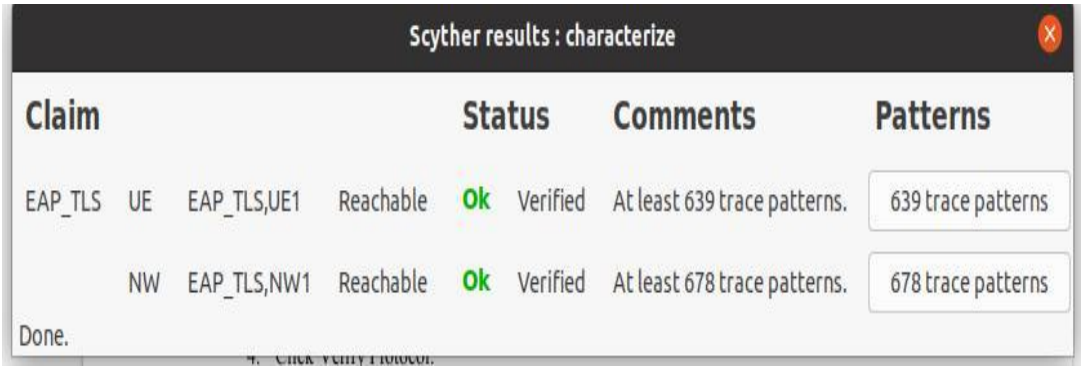
```
        claim_i4(NW,Secret,Nisynch);
    }
}
```



Scyther results : verify

Claim				Status		Comments	Patterns
EAP_TLS	UE	EAP_TLS,i1	Secret Kseaf	Ok		No attacks within bounds.	
		EAP_TLS,i2	Secret k(UE,NW)	Ok		No attacks within bounds.	
		EAP_TLS,i3	Secret Niagree	Ok		No attacks within bounds.	
		EAP_TLS,i4	Secret Nisynch	Ok		No attacks within bounds.	
	NW	EAP_TLS,NW1	Secret Kseaf	Fail	Falsified	At least 1 attack.	1 attack
		EAP_TLS,NW2	Secret k(UE,NW)	Ok		No attacks within bounds.	
		EAP_TLS,NW3	Secret Niagree	Ok		No attacks within bounds.	
		EAP_TLS,NW4	Secret Nisynch	Ok		No attacks within bounds.	

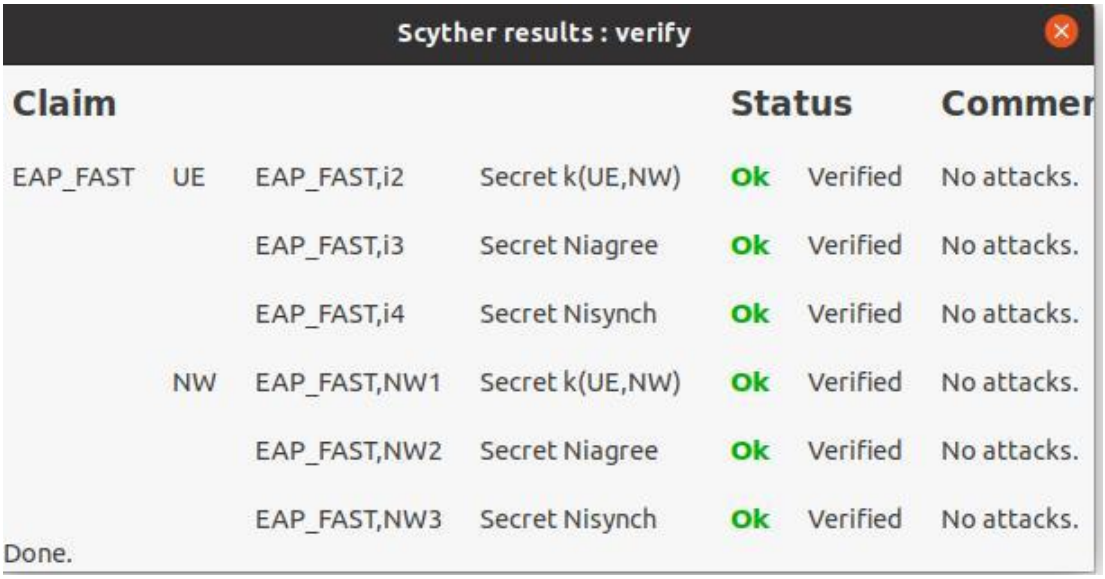
Done.

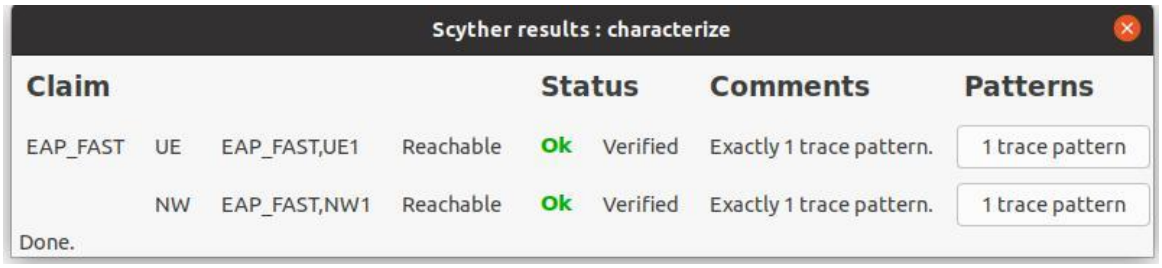


Scyther results : characterize

Claim				Status		Comments	Patterns
EAP_TLS	UE	EAP_TLS,UE1	Reachable	Ok	Verified	At least 639 trace patterns.	639 trace patterns
	NW	EAP_TLS,NW1	Reachable	Ok	Verified	At least 678 trace patterns.	678 trace patterns

Done.


Scyther results : verify
Claim
Status
Commer
EAP_FAST UE EAP_FAST,i2 Secret k(UE,NW) Ok Verified No attacks.
EAP_FAST,i3 Secret Niagree Ok Verified No attacks.
EAP_FAST,i4 Secret Nisynch Ok Verified No attacks.
NW EAP_FAST,NW1 Secret k(UE,NW) Ok Verified No attacks.
EAP_FAST,NW2 Secret Niagree Ok Verified No attacks.
EAP_FAST,NW3 Secret Nisynch Ok Verified No attacks.
Done.


Scyther results : characterize
Claim
Status
Comments
Patterns
EAP_FAST UE EAP_FAST,UE1 Reachable Ok Verified Exactly 1 trace pattern. 1 trace pattern
NW EAP_FAST,NW1 Reachable Ok Verified Exactly 1 trace pattern. 1 trace pattern
Done.

Results

The Result of running security analysis on EAP-TLS and EAP-FAST authentication protocol is down successfully using Scyther tool and is explained well in the attached demo video.