

Fall 2015

Extensible Authentication Protocol Vulnerabilities and Improvements

Akshay Baheti
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [Computer Sciences Commons](#)

Recommended Citation

Baheti, Akshay, "Extensible Authentication Protocol Vulnerabilities and Improvements" (2015). *Master's Projects*. 425.
DOI: <https://doi.org/10.31979/etd.umye-6qrp>
https://scholarworks.sjsu.edu/etd_projects/425

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Extensible Authentication Protocol Vulnerabilities and Improvements

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Akshay Baheti

Dec 2015

© 2015

Akshay Baheti

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Extensible Authentication Protocol Vulnerabilities and Improvements

by

Akshay Baheti

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Dec 2015

Thomas Austin	Department of Computer Science
---------------	--------------------------------

Mark Stamp	Department of Computer Science
------------	--------------------------------

Teodoro Cipresso	Department of Computer Science
------------------	--------------------------------

ABSTRACT

Extensible Authentication Protocol Vulnerabilities and Improvements

by Akshay Baheti

Extensible Authentication Protocol(EAP) is a widely used security protocol for Wireless networks around the world. The project examines different security issues with the EAP based protocols, the family of security protocols for Wireless LAN. The project discovers an attack on the subscriber identity module(SIM) based extension of EAP. The attack is a Denial-of-Service attack that exploits the error handling mechanism in EAP protocols. The project further proposes countermeasures for detection and a defense against the discovered attack. The discovered attack can be prevented by changing the protocol to delay the processing of protocol error messages.

ACKNOWLEDGMENTS

I am highly indebted to Dr. Thomas Austin, for his guidance and constant supervision. And also for providing necessary information regarding the project and his support in completing the project. I would like to thank my thesis committee members, Dr. Mark Stamp and Professor Teodoro Cipresso for their encouragement, insightful comments, and hard questions.

TABLE OF CONTENTS

CHAPTER

1	Introduction	1
1.1	Overview of Extensible Authentication Protocol	1
1.2	The EAP protocol family	2
1.3	Vulnerabilities in EAP based Protocols	3
2	EAP based protocols	5
2.1	Wireless Connection	5
2.2	EAP - Architecture	5
2.3	Security Issues in EAP Protocol	8
2.4	Transport Layer Security [TLS]	10
2.5	EAP-TLS Protocols	11
2.6	Weakness of EAP-TLS Protocols	12
2.7	EAP-SIM protocol	13
2.8	EAP-SIM Weakness	15
3	Implementation of Security Attack	17
3.1	Error Message Attack	17
3.2	Misleading Message Attack	21
3.3	Attack on Challenge-Response EAP Methods	22
3.4	Implementation framework	23
3.5	Optimization of the Attack	27
4	Defenses and Improvements	30

4.1	EAP Queuing Enhancement	30
4.2	EAP-SIM Protocol Improvement	31
5	Conclusion	34
 APPENDIX		
	Code Changes	37
A.1	37
A.2	37

LIST OF FIGURES

1	EAP Protocol Family	3
2	Wireless LAN Connection	6
3	EAP Authentication Stack	7
4	EAP TLS Handshake	12
5	EAP Error Message Attack 1	18
6	EAP Error Message Attack 2	20
7	EAP Mislead Message Attack	21
8	EAP SIM Messages	23
9	EAP Failure Packet	26
10	Hardware Stimulated Wireless Interfaces	26
11	EAP-SIM Connection Failing	28
12	EAP-SIM Attack success versus packets injected	29
13	EAP SIM Exchange with IPSec	32

CHAPTER 1

Introduction

Almost one fourth of the smart-phone users worldwide operates on 4G networks for internet connection, according to a study published recently in 2014 [10]. A cellphone tower supports a limited number of 3G/4G connections at any given time due to the finite spectrum that can be used at one physical location. This limitation leads to degraded service quality for customers, especially at social gathering's. Wi-Fi is one solution to this problem. Using Wi-Fi to serve cellular clients is known as 3G Wi-Fi Offloading. Wi-Fi supports a larger number of clients compared to cellular network [11]. Authentication over WiFi is a challenge though. This document reviews the EAP-SIM authentication method, which is widely used for Wi-Fi authentication and the approach to overcome one of the many possible attack on EAP-SIM. In order to understand the proposed solution, we need to review the basics of EAP, including its architecture, functionality, and its application.

1.1 Overview of Extensible Authentication Protocol

The Extensible Authentication Protocol (EAP) [2] is a standard that provides a foundation for network clients and authentication servers. EAP defines a framework that allows clients to select the authentication mechanism dynamically. The EAP mechanism to be used is determined based on the information transmitted in the Access-Request to the Server via the Remote Authentication Dial-In User Service (RADIUS) message. EAP [13] is an extension to Point-to-Point Protocol (PPP) that allows for the development of plug-ins for new authentication technologies and protocols.

Today, the two most used wireless standards, Wireless Protected Access (WPA) and Wireless Protected Access 2 (WPA2) have adopted EAP methods as their authentication mechanisms.

1.2 The EAP protocol family

EAP is an authentication framework that defines methods for usage and transport of parameters and keying information generated by EAP methods. EAP methods here includes a large set of methods defined in RFCs and some popular vendor specific implementations. EAP only defines message formats with no specifics about the type of network or the type of client supported. Protocols supporting EAP are responsible of encapsulating EAP messages within their protocol messages. EAP is more widely used compared to other wireless security protocols. For example, WPA and WPA2, the WiFi authentication standards, have adopted one-hundred EAP types as their official authentication mechanisms. Most popular EAP methods are EAP-MD5, EAP-POTP, EAP-GTC, EAP-TLS, EAP-SIM, and EAP-AKA. Figure 1 describes the basic EAP methods.

In Figure 1, we see that there are several Transport Layer Security(TLS) based EAP protocols PEAP, EAP-TTLS and EAP-FAST. TLS is a popular protocol that is a part of many networking standards. TLS is responsible for ensuring privacy between communicating applications and it's users on the network. When a client and server communicate, TLS ensures that no third party may tamper or eavesdrop their messages. The EAP-TLS based protocols are mainly used for Wireless LAN. The whole WPA wireless authentication protocol family is based on EAP-TLS protocols.

EAP-SIM and EAP-AKA [5] are based on Challenge-Response approach unlike other EAP protocols that are based on TLS. These protocols are popular among

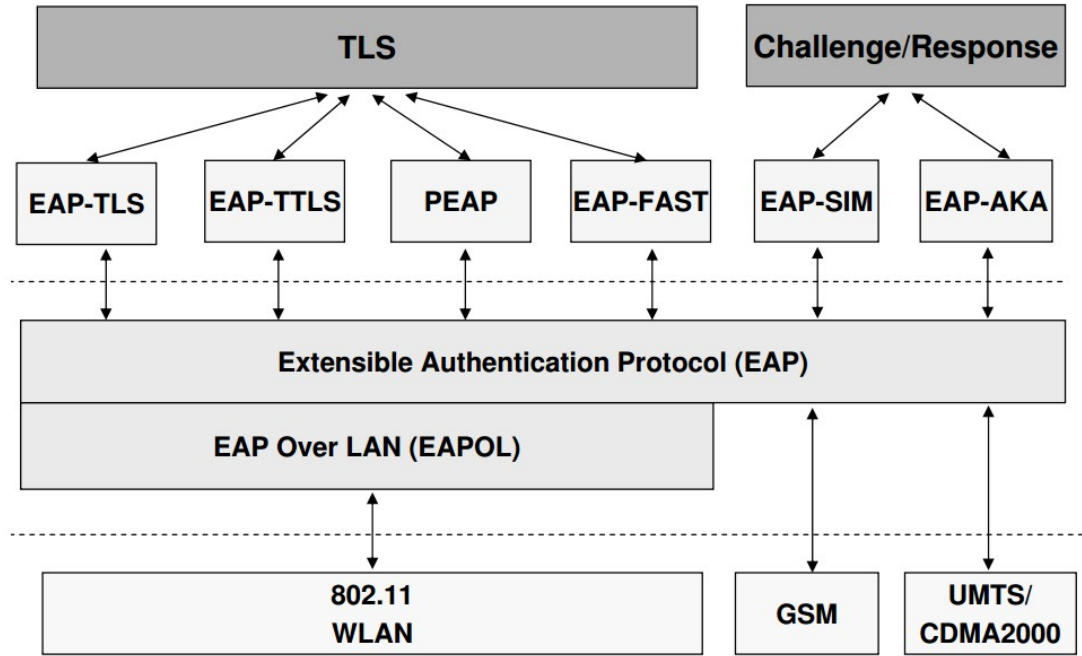


Figure 1: EAP Protocol Family

mobile users today. And the special Challenge-Response design on these protocols leads to major security flaws which we will discuss in the next chapter.

1.3 Vulnerabilities in EAP based Protocols

EAP is a protocol that is designed mainly for the authenticating wireless clients. The wireless medium has its limitations that introduce certain security threats in the protocols. The messages exchanged for establishing the session key for wireless clients and wireless routers is heard by anyone and everyone on medium. This leads to certain attacks on wireless clients. In this project, we reveal a serious vulnerability in most wireless security and communication protocols. The attack is based on sniffing the protocol communication and then injecting misleading messages and fake error messages.

The attack exploits the error handling mechanism in EAP protocols. When a spoofed EAP error message is injected by the attacker the client resets the key-exchange. This way the probability of the client connecting to the wireless router decreases. Variants of the attack can be carried out by spoofing an EAP error messages to the server or to client at various points in the authentication process. The EAP error based attack can be used on EAP-TLS based protocols. A similar attack using the SIM-Client Alert message is used for EAP Challenge-Response based protocols. This attack is easy to launch, requiring cheap everyday hardware. The attack is efficient as it needs a small number of packets and a single wireless client. The attack is generally applicable to a variety of wireless protocols and is stealthy as it does not require jamming a wireless medium like other popular wireless attacks.

This project in general explores the attack on all EAP TLS based protocols in detail. Later looks at EAP SIM and builds an attack on the same lines as the attack for EAP-TLS. We also look at open source implementations and how the attack can be carried out using this software. In addition, we propose enhancements in the EAP protocol and the software implementing this protocol. These enhancements limit the possibility of the attack. The improvement involves delaying the processing of certain protocol error and exception messages. This allows the client and server to give priority to the actual protocol message and ignore the spoofed error message thus preventing the attack. The project later suggest improvements in the wireless drivers to avoid such attacks.

CHAPTER 2

EAP based protocols

2.1 Wireless Connection

EAP was designed to provide a secure connection mechanism for wireless clients. To understand how EAP protocols work we must know where they fit in a wireless connection. Figure 2 describes the messages exchanged in a wireless connection. The wireless connection begins when the client sends a broadcast probe request to the access point. The access point responds to this with a probe response allowing the client to set certain wireless LAN physical layer parameters. After the client connects, it sends an authentication request. Following the authentication request the access point responds with the authentication response and also sends the EAP start message thus starting the EAP 802.1X authentication process to begin. This is where all EAP based protocols fit in. Following the 802.1X exchange the EAP key exchange is carried out. This is the final step in a EAP authentication.

2.2 EAP - Architecture

EAP is an extension of PPP to enable the development of various network access authentication methods. In PPP the authentication mechanism is chosen during the link establishment phase. While in EAP, the clients negotiate the EAP method during the connection authentication phase. The clients negotiate the specific EAP authentication scheme based on the client and server supported algorithms and ciphers when the authentication phase is reached. After the EAP method is decided mutually, EAP allows for an open-ended exchange of messages between the authenticating parties. The messages can vary based on the requirements of the network and

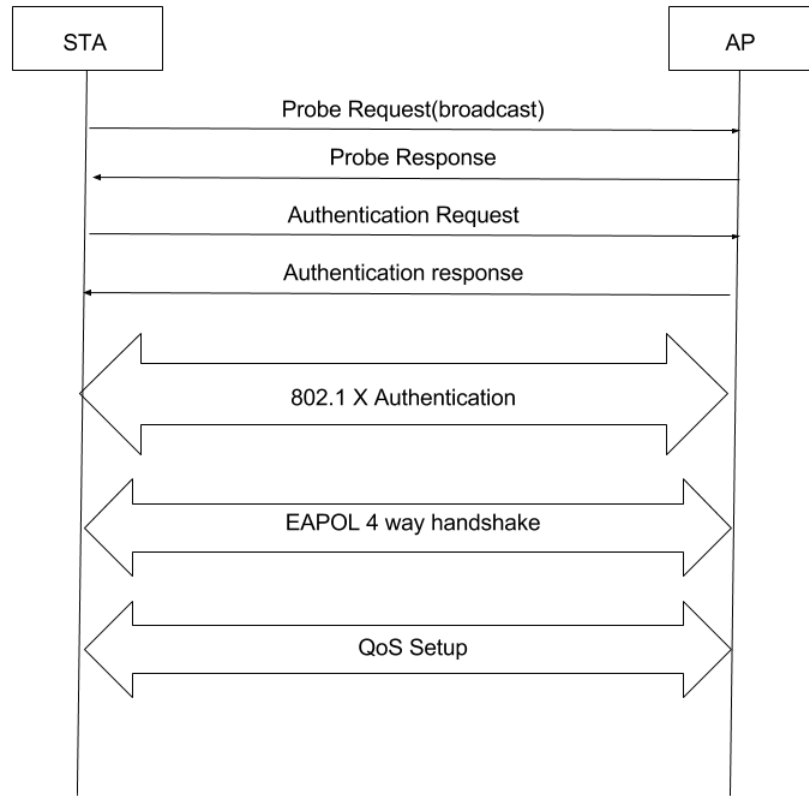


Figure 2: Wireless LAN Connection

the connection. The messages are a sequence of request and responses exchanging keying information and certificates. The EAP method determines the length and details of each authentication conversation.

EAP is a framework that is defined by the EAP methods that plug-in at both the client and the server. For EAP peers and servers to support a new EAP method the same EAP scheme library file needs to be installed at the EAP peer and the authenticating server. This ability of EAP to allow plug-in's enables vendors to create their specific new authentication schemes. Therefore EAP provides the highest flexibility as compared to other authentication schemes.

All wireless connections operate using a wireless driver specific to the physical

hardware. There are various drivers in the market. To maintain a standard for EAP methods and allow cross platform operation. The EAP mechanism is coded above the driver in userspace. All EAP clients and authentication servers send messages using a supplicant. This component authenticates clients and sends all EAP data link layer messages. All parties participating in a EAP scheme use RADIUS to send messages. The EAP authenticator and the authentication server sends EAP messages using RADIUS. Figure 3 shows EAP messages exchanged between communicating parties.

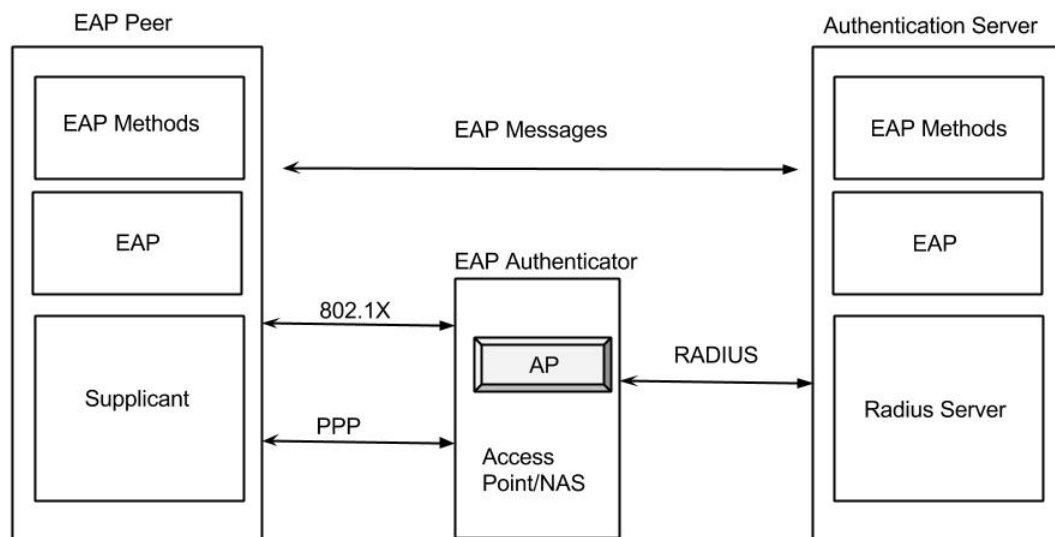


Figure 3: EAP Authentication Stack

The main components of EAP, as shown in Figure 3, are the following:

1. EAP clients - devices that support EAP authentication and trying to access the network.

2. EAP authenticator - an access point (AP) or wireless router requiring EAP authentication before granting access to the network
3. Authentication server - computer that moderates the use of a specific EAP authentication method with an EAP client. It also validates EAP peers credentials.

In Figure 3, the authentication server is a RADIUS server. EAP peer and the EAP authenticator both sends EAP messages using a supplicant and a data link layer transport protocol such as PPP or IEEE 802.1X infrastructure protocol. A supplicant is a software component that uses EAP to authenticate network access but does not handle the actual data exchange. As a result, EAP messages are actually exchanged between the EAP components on the EAP client and the authentication server. In other words, EAP provides high flexibility because it allows vendors to create more secure authentication schemes that can be plugged in later on, as required.

2.3 Security Issues in EAP Protocol

EAP is a standard that gives a framework to network access clients and authentication servers. EAP does not indicate the authentication system itself but rather a framework for custom security protocols. Since EAP does not define details of message and keys exchanged it is vulnerable to security attacks. The issues mentioned below are probably the most widely recognized security issues related to the diverse EAP usage:

1. Dictionary Attacks: A dictionary attack is a method for breaking a code or authentication component by attempting each word from a dictionary - a rundown of normal words - furthermore, encoding it the same way the first passphrase

was encoded. Dictionary attacks contrast from brute-force attacks as in a brute-force attack all probability words are attempted. A few EAP usage are powerless against dictionary attacks. For example, Cisco's Lightweight EAP (LEAP) the security depends on a shared secret, which is client's logon password. Frameworks lacking solid secret key arrangements are most susceptible to dictionary attacks. To defend against this attack, Cisco created EAP-FAST to give better assurance against dictionary attacks.

2. Plaintext Attacks: EAP executions that depend on clear-text authentication utilizing RADIUS (even inside of a secured passage) are helpless against known-plain content attacks. In a known plaintext attack (KPA), the attacker uses tests of both the plaintext and its encoded rendition to uncover further mystery data, for example, the secret encryption key. EAP-IKE2 and EAP-TTLS are examples of EAP implementations that may utilize secret word based authentication (PAP) and subsequently are vulnerable against this sort of attacks. In PAP-based authentication, passwords are transmitted decoded.
3. Ciphertext Attacks: Hypothetically, EAP-SIM enhances the original GSM security mode - based on a pre shared key and challenge-response mechanism. The initial GSM standard uses A5/1 and A5/2 stream ciphers with 64 bits as the size of key. EAP-SIM enhances the original GSM standard by expanding the key length to 128 bits. Sadly, the way the new 128-bit key is produced has been demonstrated to be blemished. Instead of being 128-bit long, the subsequent keys are 64 bits long. This design issue increases the probability to decrypt the cipher text into plain text using a key, this lowers the time complexity of the attack; which means that less time is needed for the attacker to get the necessary information.

4. Man-in-the-middle Attacks: A MitM attack is the most common attack in wireless medium as the medium allows any device to eavesdrop on the communication. The attack we discuss later is a type of MitM attack. Original implementation of EAP was based on protocols that were vulnerable to MitM attacks. In a MitM attack on a wireless network, a rogue device can act as both, the client and the server. It can spoof the communication between the parties to gain of the network. The main reasons a protocol is susceptible to MitM are
- (a) Capturing client packets and spoofing those packets to the server to act as the client. Hence gaining false access to the network.
 - (b) Clients cannot or do not properly authenticate the server, notwithstanding when the authentication protocol is utilized inside of a server-validated passage.

2.4 Transport Layer Security [TLS]

Transport Layer Security (TLS) is a cryptographic convention that gives secure correspondence on the Internet for information exchange. The convention permits client/server applications to communicate in a manner that is intended to avert the attacker from listening stealthily and fabricating messages. For instance, HTTPS convention layers on top of TLS convention to secure system traffic. TLS is comprised of the Record, the Alert, and the Handshake convention. The Record convention is intended to serve the Handshake convention and Alert protocol, and offers symmetric encryption, data authenticity, and optionally compression. In our attack we essentially assault the handshake protocol by setting off the authenticating parties with a Alert protocol. Figure 4 demonstrates the flowchart of a fruitful TLS handshake process. For the most part, the TLS server begins the technique and the client reacts

with a welcome message. The server then sends its certificate and chooses a cipher suite. Note that TLS gives a choice to verify the client by asking for the client certificate. The client returns chosen cipher, its certificate and other cryptographic data. Note that in Figure 4 we put a few messages close together to simplify the handshake showing only the important group of messages.

2.5 EAP-TLS Protocols

EAP-TLS, is the standard that uses the Transport Layer Security (TLS) protocol, and is supported among many vendors as a WLAN authentication protocol. EAP-TLS is considered to be one of the most secure EAP standards available today, although TLS provides security only if the user can understand warnings about false credentials. EAP-TLS is implemented as client and server protocol by Apple, Cisco, HP, Juniper, Microsoft, and other open source operating systems. It is usually supported in the latest versions of Mac OS, wpa_supplicant, Windows and Apple's iOS mobile operating system.

Even though the standard does not mandate the use of client-side X.509 certificates, almost all of its implementations require them to be installed. The necessity for a client-side certificate, however disliked it might be, is the thing that gives EAP-TLS its authentication quality and shows the exemplary comfort versus security trade off. With a customer side authentication, a compromised password is insufficient to break into EAP-TLS empowered frameworks in light of the fact that the intruder still needs to have the client-side certificate; for sure, a password is not by any means required, as it is just used to encode the client-side certificate for storage. The most astounding security is the point at which the "private keys" of client-side certificates are housed in smart cards. This is on the grounds that there is no real way to take a client-side

certificate related to a private key from a smart card without taking the card itself. It is more probable that the physical robbery of a smart card would be seen (and the smart card quickly repudiated) than for a typical password theft to be noticed.



Figure 4: EAP TLS Handshake

2.6 Weakness of EAP-TLS Protocols

The weakness of EAP protocols comes from the TLS protocol, which is broadly utilized as a part of numerous security protocols, such as the HTTPS and the TLS based EAP protocol. In this manner, all the TLS based EAP protocols e.g.(PEAP, EAP-TLS, EAP-TTLS and EAP-FAST) will be defenseless against our attack. TLS is a popular protocol among various security protocols. Many EAP protocols as build around TLS. The most popular EAP-TLS protocols are PEAP, EAP-TTLS and EAP-

FAST. The weakness of TLS protocol makes all EAP-TLS protocol vulnerable to our attack. On the other hand, the hardness of sniffing and deciphering application layer information in wired and encoded wireless systems makes such attacks extremely difficult. Henceforth, we concentrate on the EAP-TLS protocol, which is a MAC layer authentication protocol for remote systems.

An attacker sniffs the correspondence between the wireless client and the entrance point, reviewing the authentication methodology used during the handshake protocol of TLS. Note prior to encrypting data with session keys, TLS sets up the keys, all the key setting up packets are in clear-message and decoded. Activated by a few messages, the attacker injects spoofed messages to make the TLS authentication fail. The attacker has two primary ways to stimulate attacks on TLS:

1. Error Message Attack - mocking up ALERT messages to trick the client or the server to halt the authentication process and restart the transaction.
2. Misleading Message Attack - spoofing the authentication messages to the client or the server causing the parties to believe a error has occurred and restart the transaction.

2.7 EAP-SIM protocol

Given the exponential increase in the number of Global System for Mobile Communications(GSM) and wireless enabled mobile clients. The inter-networking of WLANs and GSM networks is inevitable. Third Generation Partnership Project(3GPP) has a standard HotSpot 2.0 that is based of EAP-SIM to define the inter-networking of these two networks. The integration of the two networks expands the service and quality of current GSM networks allowing them to support more

clients in a restricted area. The advantages of the GSM technology are - the roaming capability, the authentication, the subscription management and the key agreement procedure. Compared to cellular networks WLANs provide better bandwidth and processing capabilities. The EAP method for GSM is EAP-SIM.

EAP-SIM is wireless protocol based on GSM/GRPS authentication. EAP-SIM is used to authenticate mobile clients over a wireless network. EAP-SIM incorporates a few basic improvements over GSM that eliminate known security issues of the GSM authentication. EAP-SIM is improved as it uses a 128-bit key as compared to GSM authentication which uses a 64 bit key. And also EAP-SIM provides mutual authentication while the GSM network only authenticates the client. The authentication procedure to EAP-SIM consist of a mobile client, a AAA server, a wireless router and the GSM network. The mobile client first generates two keys from the master key. One of the to keys is used as the session key and second is used to generate the Message Authentication Code(mac) over the RAND parameters of the GSM triplets.

EAP-SIM is employed for authentication and session key distribution exploiting the credentials stored in the SIM. GSM cellular networks use the keys stored in the SIM card to authenticate the mobile client. Three Kc keys of the GSM authentication triplets to generate a Master Key in EAP-SIM. EAP-SIM use a SIM authentication formula between the client and an AAA server providing indirect authentication between the client and network. In EAP-SIM the communication between the SIM card and the Authentication Centre (AuC) replaces the necessity for a pre-established password between the consumer and AAA server.

2.8 EAP-SIM Weakness

EAP-SIM is designed keeping in mind the drawbacks of GSM authentication. But there are certain scuttle flaws in EAP-SIM protocol. EAP-SIM is designed around validating the MAC. For an attacker to impersonate the GSM server all he needs to do is to generate a valid keyed MAC using a K-auth key. Two of the three Kc keys and the related RAND parameters of the GSM triplets is required to create the valid MAC. The authentication triplets for the attack can be obtained by one of the following ways:

1. If the attacker gets access to the physical SIM card, he can get the GSM triplets.
2. A malicious software can installed on the mobile device. This software can access the GSM triplets and Using a virus or other malicious software, an adversary may mount an attack on the user platform in order to obtain triplets.
3. The attacker can hack into the GSM network and get the GSM triplets.
4. The communication between the access point the AAA server on the back-end is unauthenticated. The attacker can get access to the GSM tripets.

Using the mentioned techniques the attacker can impersonate a GSM mobile and easily enter a WLAN network. As the attack has gained access to the RAND parameters he can generate the three encryption keys Kc. Using these he can generate a valid hashed MAC. The calculated MAC can be used by the attacker to authenticate a client as legitimate network. The compromised triplets can be used by the adversary as long as the Ki, which is used to calculate the three keys remains the same. The key Ki can remain the same for years. This is one example of the attack. There are two other type of attacks on EAP-SIM which we will discuss in more detail.

Apart from the above mentioned weakness of EAP-SIM, there are a lot of other

issues in the protocol. For example, the mobile must send his IMSI(International Mobile Subscriber Identity) in plaintext during the authentication phase to the server. Here the identity of the user can be compromised. In addition EAP-SIM has version negotiation. This allows old mobile clients to downgrade to a older EAP-SIM version. Also all authentication message are send unencrypted allowing the attacker to spoof messages as we will see in later chapters.

CHAPTER 3

Implementation of Security Attack

The EAP protocol family is designed around establishing keys based of Cipher negotiation and certificate exchange. The TLS based and Challenge-Response based EAP methods have a similar error detection and recovery mechanism. This introduces the possibility of attacks as the wireless medium can be sniffed and spoofed by any wireless client.

There are two types of attacks on a EAP protocol[9]. Both the attacks are time critical may fluctuate in different circumstances. The following sections explains the attacks in detail.

3.1 Error Message Attack

EAP being designed for wireless networks has exception handling build into it. The error handling mechanism in EAP allows the clients and servers to send ALERT messages during the authentication phase to notify the opposite party that a error has occurred. The attacker can capture the EAP Alert messages and spoof these messages to the server. Figure 5 demonstrates how a the attacker can spoof messages as the wireless client. There are two points during the authentication phase one before the server receives the client HELLO message and other after the accepting the SERVER HELLO DONE. The attacker after sniffing any of the two messages sends a FATAL ALERT message. If the spoofed FATAL ALERT message reaches before the CLIENT HELLO message, then all messages following the CLIENT HELLO are dropped. This is because the server will only consider the alert messages and will send a failure message to the client indicating an error. At that point EAP ends with

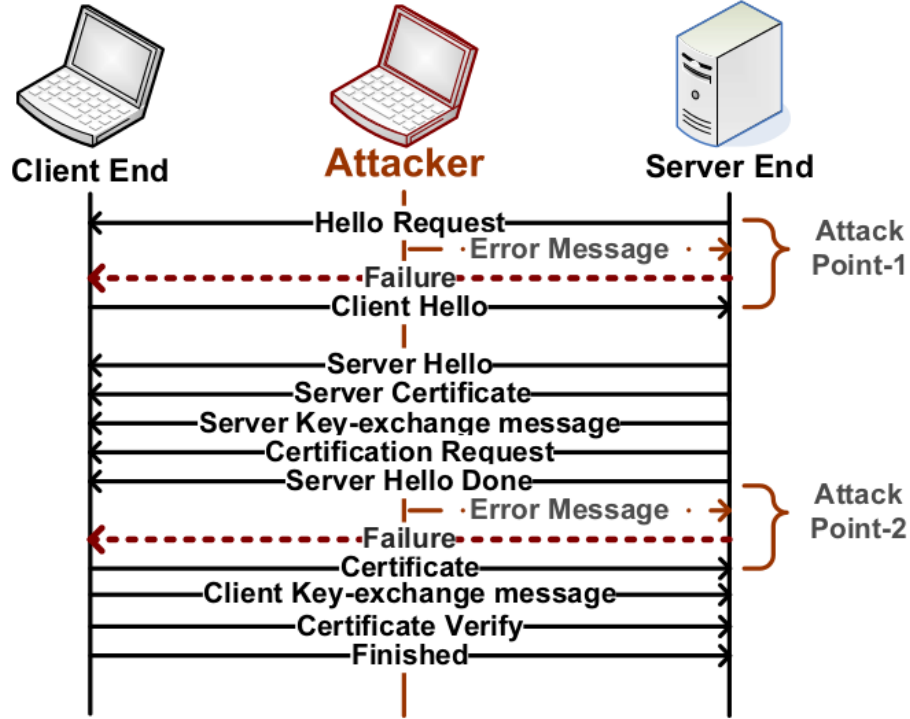


Figure 5: EAP Error Message Attack 1

an EAP FAILURE message.

A variant of the above mentioned attack can be carried out when the attacker spoofs as the server. Here the attacker sends messages to the client to trigger the error handling mechanism in the client. There are two points in the EAP authentication phase where the attacker can trigger the client. One is after the client sends the CLIENT HELLO and the other is after a set of client response messages (including CLIENT CERTIFICATE, CLIENT KEY EXCHANGE, and other messages) (See Figure 5). There are multiple attack point in the bundle of EAP packets from server to the client.

The attacks at different points in the authentication phase are similar. We will examine the first attack point in detail. As shown in figure 6, once the attacker sniffs

a CLIENT HELLO message and spoofs a FATAL ALERT message to the client. The SERVER HELLO and future messages from the server are dropped. Once the client accepts the FATAL ALERT message. It believes that a error has occurred and sends a error message to the server. Both parties now terminate the transaction and restart the EAP authentication procedure. The critical part of the attack here is to meet the timing requirement. The spoofed packet from the attacker must reach the client before the original packer from the server. This time gap contains :

1. The time required to deliver the message on the wired system, which includes messages from the TLS server to the AP and from the AP to the TLS server.
2. Time taken by the server to process the request
3. The time required to deliver messages over the wireless medium from the AP to the client.

The interval varies for different wireless networks and different types of wireless clients. It also depends on the stage of the authentication process. The second attack point as shown in Figure 6 requires the server to query the database for user identity and password. This increases the time gap for the attacker to send the spoofed packet as the response from the server will be delayed than usual. And also wireless is a open medium the transfer speed of packets also depends on the background traffic. If there are many clients in the network the wireless protocol drops the transmit speed significantly to insure reliable delivery. The attackers spoofing time is divided into two parts.

1. The time required to produce the spoofed message, which can be ignored and
2. The transmitting time in the wireless network.

A variant of the attack exists where the attacker can spoof the client. This variant of the attack also has two different attack points. As shown in Figure 6, the first attack point is right after the attackers sniffing the client HELLO REQUEST and spoofing sending a FATAL ALERT to the server. Similarly the second attack point is when the client receives the SERVER HELLO message.

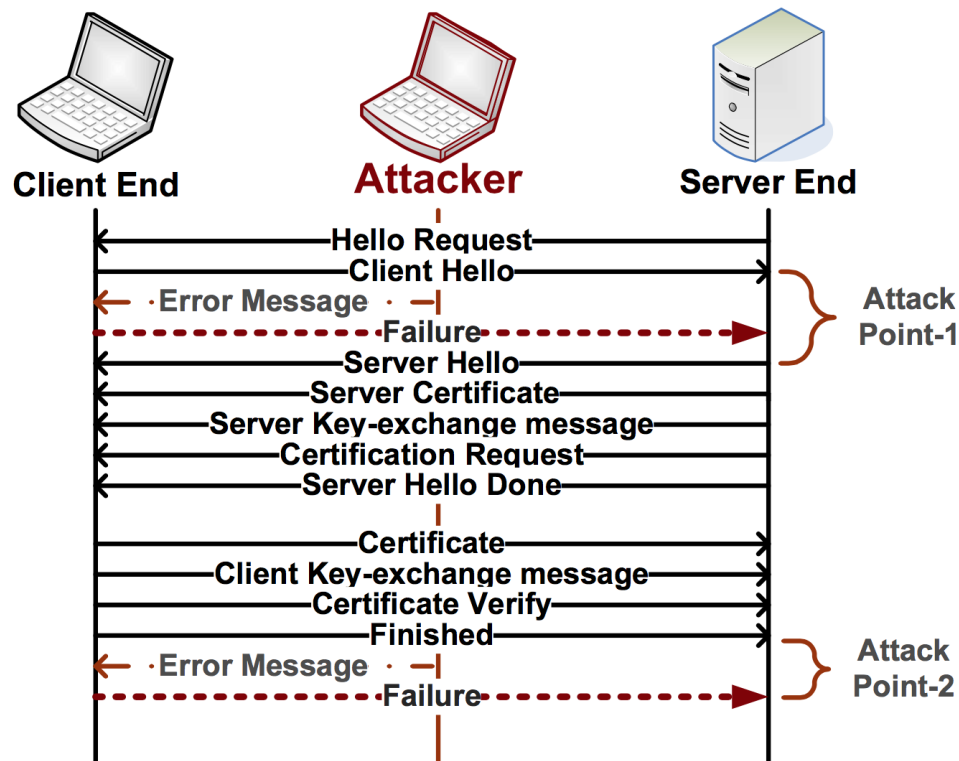


Figure 6: EAP Error Message Attack 2

Now let's look at a variant of the first attack. Here we spoof as the client to the server. As shown in Figure 6, the attacker's FATAL ALERT message must reach the server before the CLIENT HELLO message. All messages from the client now are dropped by the server as it believes an error has occurred. The server now ends the transaction by sending a failure message. The second attack point is quite similar where the attacker sends the FATAL ALERT message before the client certificate.

reaches the server.

3.2 Misleading Message Attack

The approach to break EAP is by sending misleading messages to the client. Since the client has no way to verify the messages in the initial phase of EAP authentication. The attacker can easily spoof messages to the client. The attack can be carried out using a spoofed SERVER HELLO message. This HELLO message will have misconfigured parameters. These parameters will throw off the client forcing it to send a failure message.

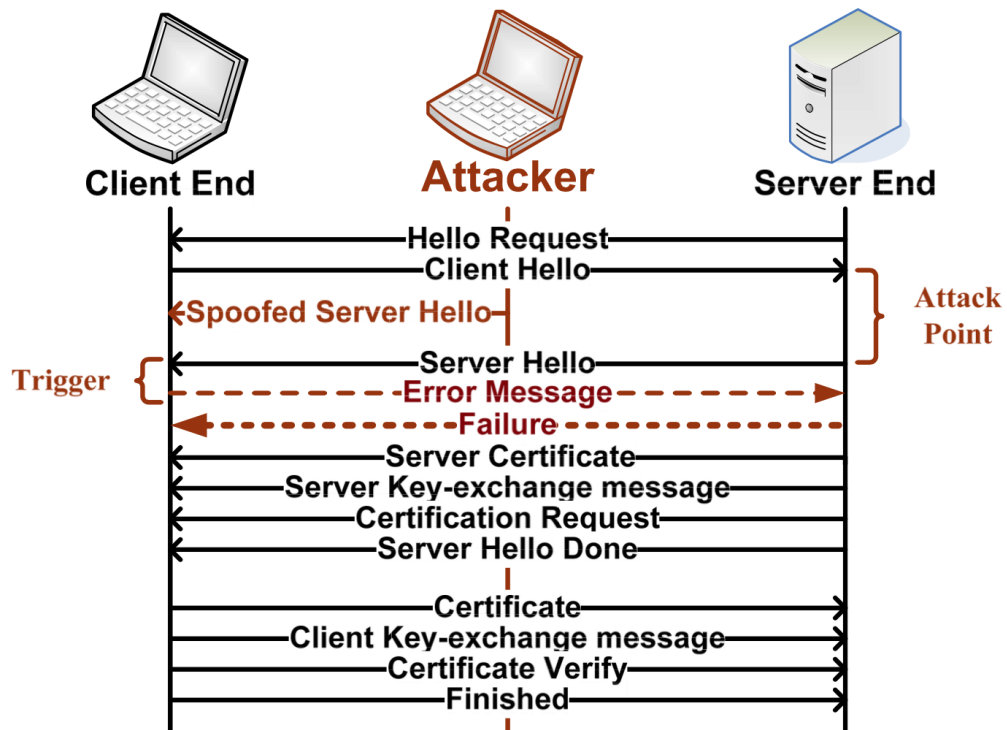


Figure 7: EAP Mislead Message Attack

As shown in the Figure 7 first the client sends a valid CLIENT HELLO message with the list of supported cipher suites. The server selects a cipher suite from the list sent by the client. Beside his choice the server must also send a set of important

keying messages to the client for the client to respond. Here the attacker can spoof a SERVER-HELLO message to the client. Now the client receives one original SERVER HELLO message and one spoofed message. Here the client is not able to decide which of the two SERVER HELLO messages is the legitimate one that should be used in the following process. Moreover the spoofed message may have selected cipher that was not listed by the client. This will result in the client sending a failure message. The failure message notifies the server side that an error has occurred, eventually terminates the whole handshake procedure.

3.3 Attack on Challenge-Response EAP Methods

While working on the EAP implementation based Challenge/Response approach, the foundation of the EAP-SIM method, we encountered a possibility of an attack. This attack comes into existence at different scenarios and at different time instances of the Challenge/Response handshake.

We will first look at the messages exchanged in a EAP-SIM protocol in Figure 8. The EAP-SIM client is wireless(WLAN) client with a SIM(subscriber identity module) to allow authentication. When a Client tries to initiate a wireless connection with a router. It first carries out the wireless Layer 2 connection. Following the connection the Server send a EAP-Request. This requires the client to respond with the supported ciphers suites and a response to a challenge sent in the request. Once the EAP-Response is received the server sends a SIM-Challenge. The client using the SIM Keys responds to the SIM-Challenge. Between the SIM-Challenge and the SIM-Response a attacker can inject a spoofed SIM-Client Error. Seeing the client error the server believes the client rejected the authentication and restarts the EAP exchange when the client reconnects. The attacker when sending the authentication reject to

the server at the same time sends a spoofed SIM-Notification to the client. When the client receives the notification it believes the server ended the authentication midway and reconnects to the wireless router. Figure 8 demonstrates the attack on EAP-SIM.

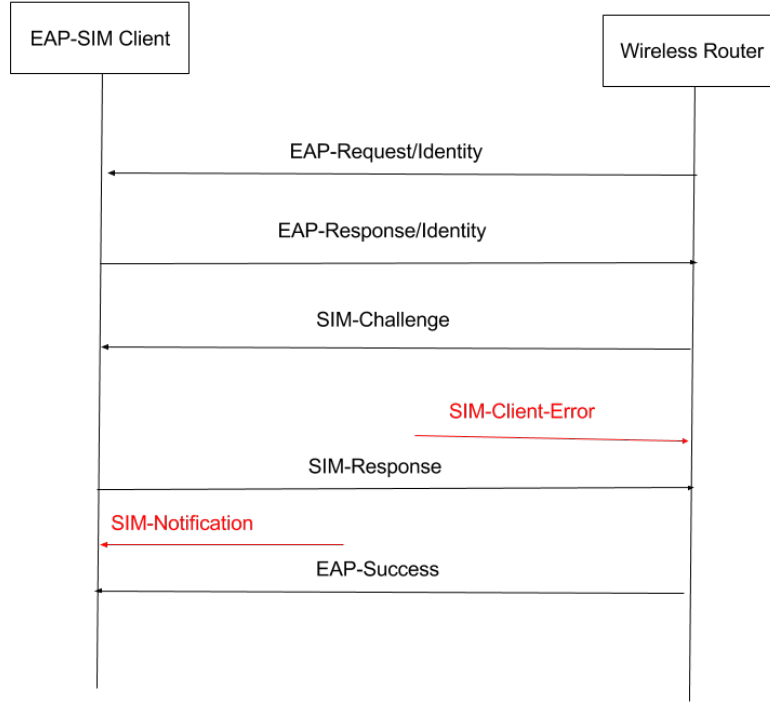


Figure 8: EAP SIM Messages

3.4 Implementation framework

In this section, we discuss our attack framework. The following are a few requirements to execute the attack.

Media Requirements: Our attacks are on the wireless medium. This makes sniffing and spoofing an integral part of our attack. Sniffing allows us to estimate the stage the protocol is in and to decide when to inject the attack packets. Modern day switches easily detect sniffing in Ethernet and it is almost impossible to sniff on enterprise

grade routers. In wired networks today enterprises use firewalls to block IP spoofing. On the other hand in wireless networks the user has no control over the medium. This allows the attack to sniff and spoof the network easily with free software and consumer hardware. This makes TLS based protocols are so vulnerable in wireless networks. In principal the attack can work on a wired network

Protocol requirements: Usually a few control messages are exchanged by communicating parties in security protocols to establish session keys. These control messages are unencrypted and can be read by the attacker. These security protocols normally have exception handling mechanism given the unreliable nature of the wireless medium. This exception handling helps inform the other party when a error has occurred. Most protocols have simple fatal or unexpected exceptions, which result in the communicatin party ending the transaction, and restarting a new handshake process. This unencrypted and open-ended exception handling makes the protocol vulnerable to DoS attack.

Timing requirements: Once a protocol meets the above requirements, the attacker can launch a DoS attack on the protocol. The attack will only succeed if the spoofed packet reach the target before the original packet. This makes the attack very time critical. The attack is feasible only if the time window is wide enough for the attacker to spoof packets.

Once all three requirements are satisfied the attack can be carried out. The attacker sniffs the medium to estimate when to initiate the attack. Then when the time is right he spoofs a message to one party, making it believe that an error has occurred. This results in that party terminating the authentication. The attack is successful if the spoofed message reached the communicating party before the original message. Otherwise, if the party receives the expected message first, it will process

that and move to a new state. The attack message now is obsolete and is discarded silently. Hence, the attack must happen after the attacker receives the trigger message and finish before the expected message reaches the client.

In the experiment setup we use a wireless stimulator[15] to execute the attack. The setup consist of the following components:

1. Hardware stimulator(hwsim) driver
2. Wireless client open source software(wpa_supplicant)
3. Wireless server open source software(hostapd)
4. Script for EAP methods connections
5. Script to inject spoofed packets

Before we can carryout the attack, we need to capture an EAP FATAL ALERT packet. This can be done by trying out a EAP authentication with a wireless router and changing the operating channel of the client when the authentication is going on. This will result in EAP FATAL ALERT from the server to the client. This packet can be capture by using Wireshark. Figure 9 shows a screen shot of a capture EAP failure packet. This packet was used to carry on the attack.

Once we have captured the EAP failure packet. We can setup our wireless stimulation environment. This is done using a start script that inserts the hardware stimulation driver module and does ioctl(input/output control) for creating 3 wireless interfaces. These interfaces help us stimulate a wireless environment. The advantage of using a stimulator is we eliminate any loss that might happen over a real wireless

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	02:00:00:00:03:00	02:00:00:00:00:00	EAP	22	Failure

Frame 1: 22 bytes on wire (176 bits), 22 bytes captured (176 bits)	
▼	Ethernet II, Src: 02:00:00:00:03:00 (02:00:00:00:03:00), Dst: 02:00:00:00:00:00 (02:00:00:00:00:00)
▶	Destination: 02:00:00:00:00:00 (02:00:00:00:00:00)
▶	Source: 02:00:00:00:03:00 (02:00:00:00:03:00)
	Type: 802.1X Authentication (0x888e)
▼	802.1X Authentication
	Version: 802.1X-2004 (2)
	Type: EAP Packet (0)
	Length: 4
▼	Extensible Authentication Protocol
	Code: Failure (4)
	Id: 47
	Length: 4

0000	02 00 00 00 00 00 02 00 00 00 03 00 88 8e 02 00
0010	00 04 04 2f 00 04

Figure 9: EAP Failure Packet

network and the probability of the attack succeeding increases. Figure 10 shows the 3 wireless interface that we setup.

wlan0	Link encap:Ethernet HWaddr 02:00:00:00:00:00 inet6 addr: fe80::ff:fe00:0/64 Scope:Link UP BROADCAST MULTICAST MTU:1500 Metric:1 RX packets:39 errors:0 dropped:0 overruns:0 frame:0 TX packets:31 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:5818 (5.8 KB) TX bytes:5307 (5.3 KB)
wlan1	Link encap:Ethernet HWaddr 02:00:00:00:01:00 inet6 addr: fe80::ff:fe00:100/64 Scope:Link UP BROADCAST MULTICAST MTU:1500 Metric:1 RX packets:7 errors:0 dropped:0 overruns:0 frame:0 TX packets:7 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:667 (667.0 B) TX bytes:711 (711.0 B)
wlan2	Link encap:Ethernet HWaddr 02:00:00:00:02:00 inet6 addr: fe80::ff:fe00:200/64 Scope:Link UP BROADCAST MULTICAST MTU:1500 Metric:1 RX packets:4 errors:0 dropped:0 overruns:0 frame:0 TX packets:3 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:113 (113.0 B) TX bytes:195 (195.0 B)

Figure 10: Hardware Stimulated Wireless Interfaces

Once the stimulation environment is setup we must execute the EAP-SIM test.

This test carries out an end-to-end EAP-SIM authentication over the wireless interface. Sniffing packets on the wireless interface during the authentication, we can see the message exchange involved in an EAP-SIM authentication. Now for our attack we want to inject the EAP failure packet we captured earlier during this EAP-SIM exchange. A tcpdump command can inject the packet on the wireless interface. This results in a possibility of the EAP exchange failing for the client. The following inject script will inject the EAP failure packet.

```
#!/bin/sh
#This script uses the tcpdump command to inject the
#packets in user specified interface and interval

tcpreplay --mbps=0.001 --loop=$2 -i $1 EAP_failure_packet.pcap
```

The Figure 12 shows the probability for the attack to succeed based on the number of packets injected by the attacker. By changing the number of packets we inject we have a results showing the probability of success of the attack. When we inject around a 1000 packets we see the attack succeeds with a probability of 15 percent while when we inject 5000 packets we see that the attack almost always succeeds.

3.5 Optimization of the Attack

Attacks on EAP protocols can be successfully if the spoofed packets from the attacker reaches the client before the actual packet reaches from the server. To increase the probability of our attack we require the attackers packet to reach the victim faster. The following optimization will allow the attacker packet to reach the client earlier we do the following optimization.

```

DEV: wlan2: 02:00:00:00:02:00
APDEV: wlan3
APDEV: wlan4
wlan0: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
wlan1: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
wlan2: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
START ap_wpa2_eap_sim 1/1
Test: WPA2-Enterprise connection using EAP-SIM
Starting AP wlan3
Connect STA wlan0 to AP
Connect STA wlan1 to AP
Connect STA wlan2 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
EAP-SIM Connection Failed.
Connect STA wlan0 to AP
wlan0: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
wlan1: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
wlan2: Ignore unexpected status-driver line: FAIL-NO-IFNAME-MATCH
PASS ap_wpa2_eap_sim 0.38944 2015-11-08 00:59:11.316841
passed all 1 test case(s)
root@ubuntu:~/hostap/tests/hwsim#

```

Figure 11: EAP-SIM Connection Failing

The optimization relies on making changes in the collision avoidance mechanism used by Wireless LAN. Wireless LAN used (Distributed Coordination Function) based on CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). In CSMA/CA, a wireless transmitting device performs a arbitrary back-off before transmitting a frame over wireless. Once it finds the medium idle, the station selects a random back off period from $[0, CW - 1]$, where CW is the contention window. The contention window CW has an initial minimum value CW_{Min} . The contention window value is doubled by the station, when a collision is detected in the medium. The contention window keeps doubling up to the maximum value CW_{Max} . The contention window is reset to it's initial value when the packet is transmitted. The value

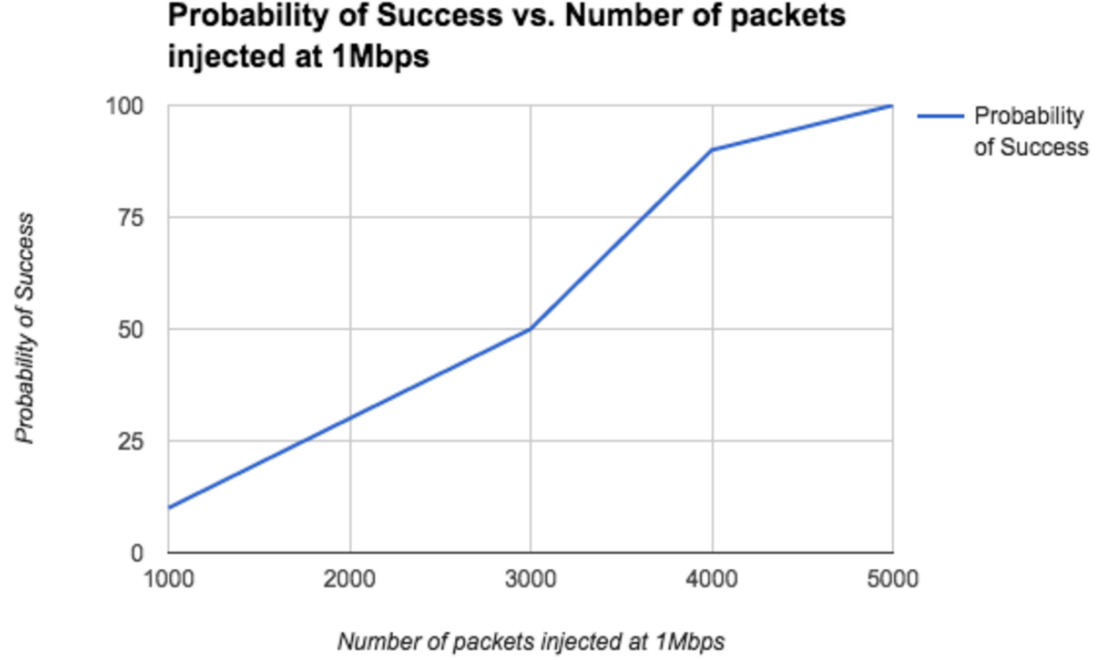


Figure 12: EAP-SIM Attack success versus packets injected

of CWMin and CWMax can be configured in open source wireless software. To send packet faster, a attacker can fix the CWMin and CWMax to the minimal number of 1. This way the back-off period for the attacker is smaller as compared to that of the other wireless stations. Hence there is high probability of the attacker packet reaching the victim before the legitimate packet. In MADWifi driver [12] provides ioctl's to set the CWMin and CWMax.

CHAPTER 4

Defenses and Improvements

The main reason we have the security threats is that wireless is an uncontrolled medium. Anyone can capture your communication and spoof/replay your messages causing problems. It is difficult for the receiving party to determine the source of the sender, especially during the initial key generation phase of communication. That is the time when clients are trying to authenticate each other. Since wireless is unreliable median we need a well-structured error handling and error recovery mechanism, for the wireless protocol to work reliably. Achieving both strong authentication and error recovery is a challenge in the wireless medium.

4.1 EAP Queuing Enhancement

In the Error message based attack, the attacker spoofs the error message he captured earlier to mislead the client. A possible defense to this attack is to allow the receiving party to distinguish between the spoofed error message and the original error message. Given the nature of the medium this is difficult to achieve especially during the authentication stage, since that is when we are trying to authenticate and verify the clients. Queueing the error message in the receiving party is a possible approach. Delaying processing of the FATAL ALERT message does affect the working EAP protocol. The wireless client agent `wpa_supplicant` and server agent `hostapd` can be configured to delay the processing of any FATAL ALERT message by 100 milliseconds. This prevents the handshake from failing due to spoofed error messages.

4.2 EAP-SIM Protocol Improvement

The identified shortcomings of the EAP-SIM authentication (in section 2.8) procedure, threatens the security of end-clients and 3G-WLAN networks. To overcome these security issues, a improved EAP-SIM authentication protocol is proposed. The new scheme uses a IPsec-based VPN deployment over the wireless network. This proposed improvement helps secure the initial set of unauthenticated messages by encapsulating them in a VPN tunnel.

IP Security Protocol (IPsec) is designed for enabling mutual authentication while maintaining a secure association between communication parties. Internet Key Exchange (IKEv2) [6] used to secure message is a part of IPsec. IKEv2 is used as it supports all legacy authentication methods, which includes all EAP based protocols. This allows us to encapsulate EAP-SIM messages in IKE payloads. Hence protecting them during negotiation. The EAP-SIM protocol generates an MSK key, which is later used to authenticate peers in the IKE procedure. An IPsec-SA is established after the EAP-SIM authentication is complete. This VPN tunnel now protects all user data.

The proposed scheme is more relevant in context to a 3G-WLAN network. A network that enables 3G clients to connect to Wireless LAN network. A 3G-WLAN network has a mobile user, a Network Access Server(NAS), GSM network, a Certificate Authority(CA) and an AAA server. The client is a IKE client and a EAP client at the same time. The NAS incorporates VPN capabilities and also replies to IKE messages from the mobile client. The client must verify the certificate from the NAS with the CA. Also the client communicates its Radius messages to the AAA Server via the NAS. After receiving the Radius message from the client AAA server is responsible of communicating with the GSM server to obtain the GSM triplets.

These GSM triplets help NAS validate the SIM keys on the GSM client.

IKE is an application layer protocol while EAP-SIM is a layer two protocol. Hence a mobile user must use EAP-SIM initial phase to obtain a temporary IP address. This IP address is only used in the authentication process until the user is not assigned a permanent IP address by the VPN. The permanent address is allowed the user to access the network. This IP address is exchanged as an IKE payload in an encrypted form.

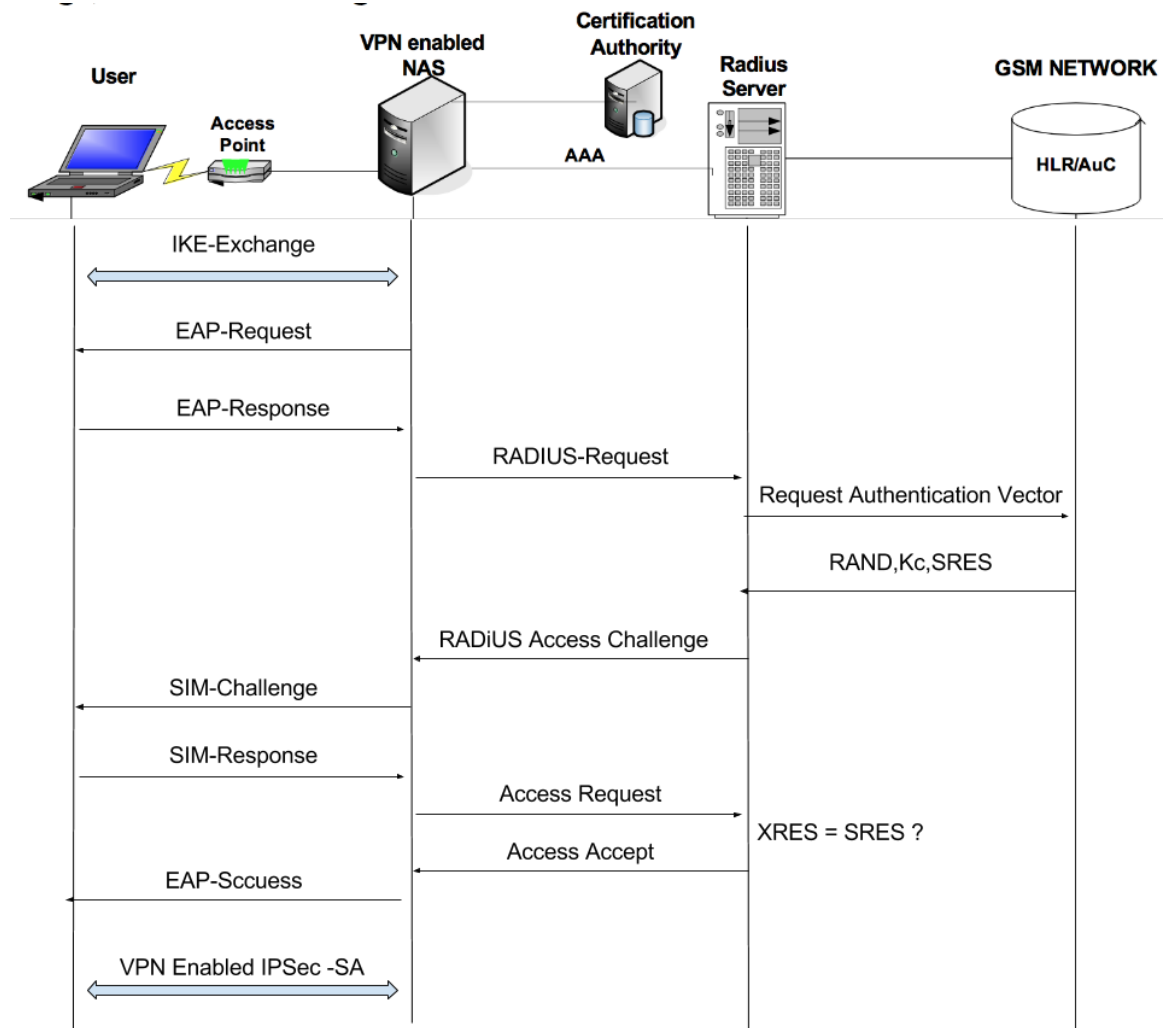


Figure 13: EAP SIM Exchange with IPSec

Figure 13 describes how IKEv2 can be used to improve EAP-SIM[7]. First an

IKE key exchange is carried out via the Access Point. This exchange consist of 2 phases IKE_SA INIT and IKE_AUTH. Once the IKE exchange is done the EAP SIM key exchange starts. Now the client and the RADIUS server exchange the RADIUS Access-Request and RADIUS Access-Challenge. This leads to the SIM Start message from client to the Radius Server. The Radius Server now in the background communicates with the Home Location Register(HLR) in the GSM network to get the SRES. When the mobile user responds with the XRES to the SIM Start message. The RADIUS server now compares the XRES and SRES to verify the SIM credentials. Once the SIM keys are verified, the two parties are mutually authenticated. Now the IPsec tunnel can be setup.

The current NAS does store previously used RANDs. To prevent a replay attack the NAS must maintain a list of all previous used RANDs. This does not allow the client to reuse the RANDs. But since the client does not store the RANDs it cannot verify the RANDs used by the server. The new EAP-SIM scheme can be further improved by storing all previously used RANDs on the NAS Server.

CHAPTER 5

Conclusion

In this paper, we study the EAP based protocols and propose attacks on the same. The attacks explore the wireless nature of the medium and the ability to spoof exchanged messages. Stimulation experiments using open source wireless stimulator in the project prove the success and the practicality of the attacks. In the experiments we spoof exception message to the client and server to carry out a Denial-of-Service attack.

We also propose an improvement in client and server software to prevent such attacks. The improvement involves delaying the processing of protocol error messages. As future work we could implement the discussed VPN based improvements for all generic wireless authentication protocols based on EAP.

LIST OF REFERENCES

- [1] C.deLaat, L.Gommans, G.Gross, J.Vollbrecht, D.Spence, "Generic AAA Architecture", RFC 2903, Aug 2000.
- [2] B. Aboba, E. H. Lev, J. Vollbrecht, J. Carlson, and L. Blunk, Extensible Authentication Protocol (EAP). RFC 3748, Jun. 2004.
- [3] H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, Feb 1997.
- [4] H. Haverinen, "EAP-SIM Authentication" Internet draft, draft-haverinen-pppext-eap-sim-16, Dec 2004
- [5] J. Arkko and H. Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187, Jan. 2006.
- [6] C.Kaufman, "Internet key exchange (IKEv2) protocol" Internet draft, draft-ietf-ipsec-ikev2-17.txt, Sep 2004.
- [7] Christoforos Ntantogian and Christos Xenakis "An Enhanced EAP-SIM Authentication Scheme for Securing WLAN" , Mobile-Summit, 2006
- [8] D. Harkins and D. Carrel. "The Internet key exchange (IKE)", RFC 2409, Nov 1998.
- [9] Yao Zhao, Sagar Vemuri, Jiazhen Chen, Yan Chen and Hai Zhou "Exception Triggered DoS Attacks on Wireless Networks", Motorola Labs, 2008
- [10] "Growth of mobiles on 4G Network" Retrieved 10, December 2014, from <http://vision.visaeurope.com/1-out-of-4-mobile-phone-users-is-on-a-4g-network/>
- [11] "Growing Wireless and GSM Clients" Retrieved on 29, November 2014, from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.2157>
- [12] "Madwifi OpenSource Driver" Retrieved on 29, March 2015, from <http://madwifi.org/>.
- [13] "Extensible Authentication Protocol Overview" Retrieved on 10, December 2014, from <https://technet.microsoft.com/en-us/library/bb457039.aspx>
- [14] "EAP-TLS Wikipedia" Retrieved on 10, January 2015, from <https://en.m.wikipedia.org/wiki/EAP-TLS#EAP-TLS>

- [15] "HWSIM Wireless Network Stimulator" Retrieved on 20, May 2015, from https://wireless.wiki.kernel.org/en/users/drivers/mac80211_hwsim
- [16] "Working of EAP PEAP and EAP-TTLS" Retrieved on 25, January 2015, from <http://www.tech-faq.com/eap-leap-peap-and-eap-tls-and-eap-ttls.html>
- [17] "WPA Supplicant" Retrieved on 10, February 2015, from http://hostap.epitest.fi/wpa_supplicant/

APPENDIX

Code Changes

A.1

This code is a part of the defence to delay processing of error packets in the client agent.

```
wpa_dbg(wpa_s, MSG_DEBUG, "Not_associated_-_Delay_processing_
↳ "
        "of_received_EAPOL_Error_frame_(state=%s_bssid="
        ↳ MACSTR ")",
        wpa_suppliment_state_txt(wpa_s->wpa_state),
        MAC2STR(wpa_s->bssid));
wpabuf_free(wpa_s->pending_eapol_rx);
wpa_s->pending_eapol_rx = wpabuf_alloc_copy(buf, len);
if (wpa_s->pending_eapol_rx) {
    os_get_reftime(&wpa_s->pending_eapol_rx_time);
    os_memcpy(wpa_s->pending_eapol_rx_src, src_addr,
              ETH_ALEN);
}
```

A.2

These are the python test cases **for** EAP-SIM. These are

```
↳ stimulated test run in the mac80211 environment.
def test_ap_wpa2_eap_sim(dev, apdev):
    """WPA2-Enterprise_connection_using_EAP-SIM"""
```

```

check_hlr_auc_gw_support()
params = hostapd.wpa2_eap_params(ssid="test-wpa2-eap")
hapd = hostapd.add_ap(apdev[0]['ifname'], params)
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="90dca4eda45b53cf0f12d7c9c3bc6a89:
            ↪ cb9cccc4b9258e6dca4760379fb82581")
hwsim_utils.test_connectivity(dev[0], hapd)
eap_reauth(dev[0], "SIM")

eap_connect(dev[1], apdev[0], "SIM", "1232010000000001",
            password="90dca4eda45b53cf0f12d7c9c3bc6a89:
            ↪ cb9cccc4b9258e6dca4760379fb82581")
eap_connect(dev[2], apdev[0], "SIM", "1232010000000002",
            password="90dca4eda45b53cf0f12d7c9c3bc6a89:
            ↪ cb9cccc4b9258e6dca4760379fb82581",
            expect_failure=True)

logger.info("Negative_test_with_incorrect_key")
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="ffdca4eda45b53cf0f12d7c9c3bc6a89:
            ↪ cb9cccc4b9258e6dca4760379fb82581",
            expect_failure=True)

logger.info("Invalid_GSM-Milenage_key")

```

```
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="ffdca4eda45b53cf0f12d7c9c3bc6a",
            expect_failure=True)
```

```
logger.info("Invalid_GSM-Milenage_key(2)")
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="ffdca4eda45b53cf0f12d7c9c3bc6a8q:
            ↪ cb9cccc4b9258e6dca4760379fb82581",
            expect_failure=True)
```

```
logger.info("Invalid_GSM-Milenage_key(3)")
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="ffdca4eda45b53cf0f12d7c9c3bc6a89:
            ↪ cb9cccc4b9258e6dca4760379fb8258q",
            expect_failure=True)
```

```
logger.info("Invalid_GSM-Milenage_key(4)")
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="
            ↪ ffdca4eda45b53cf0f12d7c9c3bc6a89qcb9cccc4b9258e6dc
            ↪ ",
```



```

        expect_failure=True)

logger.info("Missing_key_configuration")
dev[0].request("REMOVE_NETWORK_all")
eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            expect_failure=True)

def test_ap_wpa2_eap_sim_sql(dev, apdev, params):
    """WPA2-Enterprise_connection_using_EAP-SIM_(SQL)"""
    check_hlr_auc_gw_support()
    try:
        import sqlite3
    except ImportError:
        raise HwsimSkip("No_sqlite3_module_available")
    con = sqlite3.connect(os.path.join(params['logdir'], "
        ↪ hostapd.db"))
    params = hostapd.wpa2_eap_params(ssid="test-wpa2-eap")
    params['auth_server_port'] = "1814"
    hostapd.add_ap(apdev[0]['ifname'], params)
    eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
                password="90dca4eda45b53cf0f12d7c9c3bc6a89:
                ↪ cb9cccc4b9258e6dca4760379fb82581")

    logger.info("SIM_fast_re-authentication")
    eap_reauth(dev[0], "SIM")

```



```

eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="90dca4eda45b53cf0f12d7c9c3bc6a89:
                ↪ cb9cccc4b9258e6dca4760379fb82581")

with con:
    cur = con.cursor()
    cur.execute("UPDATE_reauth_SET_counter='10'_WHERE_
                ↪ permanent='1232010000000000'")
eap_reauth(dev[0], "SIM")
with con:
    cur = con.cursor()
    cur.execute("UPDATE_reauth_SET_counter='10'_WHERE_
                ↪ permanent='1232010000000000'")
logger.info("SIM_reauth_with_mismatching_counter")
eap_reauth(dev[0], "SIM")
dev[0].request("REMOVE_NETWORK_all")

eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
            password="90dca4eda45b53cf0f12d7c9c3bc6a89:
                ↪ cb9cccc4b9258e6dca4760379fb82581")

with con:
    cur = con.cursor()
    cur.execute("UPDATE_reauth_SET_counter='1001'_WHERE_
                ↪ permanent='1232010000000000'")
logger.info("SIM_reauth_with_max_reauth_count_reached")

```

```

eap_reauth(dev[0], "SIM")

def test_ap_wpa2_eap_sim_config(dev, apdev):
    """EAP-SIM_configuration_options"""
    params = hostapd.wpa2_eap_params(ssid="test-wpa2-eap")
    hostapd.add_ap(apdev[0]['ifname'], params)
    dev[0].connect("test-wpa2-eap", key_mgmt="WPA-EAP", eap="
        ↪ SIM",
                    identity="1232010000000000",
                    password="90dca4eda45b53cf0f12d7c9c3bc6a89
                        ↪ :cb9cccc4b9258e6dca4760379fb82581",
                    phase1="sim_min_num_chal=1",
                    wait_connect=False, scan_freq="2412")
    ev = dev[0].wait_event(["EAP:_Failed_to_initialize_EAP_
        ↪ method:_vendor_0_method_18_(SIM)"], timeout=10)
    if ev is None:
        raise Exception("No_EAP_error_message_seen")
    dev[0].request("REMOVE_NETWORK_all")

    dev[0].connect("test-wpa2-eap", key_mgmt="WPA-EAP", eap="
        ↪ SIM",
                    identity="1232010000000000",
                    password="90dca4eda45b53cf0f12d7c9c3bc6a89
                        ↪ :cb9cccc4b9258e6dca4760379fb82581",
                    phase1="sim_min_num_chal=4",

```

```

        wait_connect=False , scan_freq="2412")
ev = dev[0].wait_event(["EAP:_Failed_to_initialize_EAP_
    ↪ method:_vendor_0_method_18_(SIM)"], timeout=10)
if ev is None:
    raise Exception("No_EAP_error_message_seen_(2)")
dev[0].request("REMOVE_NETWORK_all")

eap_connect(dev[0], apdev[0], "SIM", "1232010000000000",
    password="90dca4eda45b53cf0f12d7c9c3bc6a89:
    ↪ cb9cccc4b9258e6dca4760379fb82581",
    phase1="sim_min_num_chal=2")
eap_connect(dev[1], apdev[0], "SIM", "1232010000000000",
    password="90dca4eda45b53cf0f12d7c9c3bc6a89:
    ↪ cb9cccc4b9258e6dca4760379fb82581",
    anonymous_identity="345678")

def test_ap_wpa2_eap_sim_ext(dev, apdev):
    """WPA2-Enterprise_connection_using_EAP-SIM_and_external_
    ↪ GSM_auth"""
    try:
        _test_ap_wpa2_eap_sim_ext(dev, apdev)
    finally:
        dev[0].request("SET_external_sim_0")

```

Other supporting functions

```

def eap_connect(dev, ap, method, identity, anonymous_identity
    ↪ =None,

        password=None,
        phase1=None, phase2=None, ca_cert=None,
        domain_suffix_match=None, password_hex=None,
        client_cert=None, private_key=None, sha256=
            ↪ False,
        fragment_size=None, expect_failure=False,
        local_error_report=False,
        ca_cert2=None, client_cert2=None,
            ↪ private_key2=None):
    hapd = hostapd.Hostapd(ap[ 'ifname' ])
    id = dev.connect("test-wpa2-eap", key_mgmt="WPA-EAP_WPA-
        ↪ EAP-SHA256",

        eap=method, identity=identity,
        anonymous_identity=anonymous_identity,
        password=password, phase1=phase1,
            ↪ phase2=phase2,
        ca_cert=ca_cert, domain_suffix_match=
            ↪ domain_suffix_match,
        wait_connect=False, scan_freq="2412",
        password_hex=password_hex,
        client_cert=client_cert, private_key=
            ↪ private_key,

```

```

        ieee80211w="1", fragment_size=
            ↪ fragment_size ,
        ca_cert2=ca_cert2 , client_cert2=
            ↪ client_cert2 ,
        private_key2=private_key2)
eap_check_auth(dev , method , True , sha256=sha256 ,
    expect_failure=expect_failure ,
    local_error_report=local_error_report)
if expect_failure:
    return id
ev = hapd.wait_event([ "AP-STA-CONNECTED" ], timeout=5)
if ev is None:
    raise Exception("No_connection_event_received_from_
        ↪ hostapd")
return id

def eap_check_auth(dev , method , initial , rsn=True , sha256=
    ↪ False ,
    expect_failure=False , local_error_report=
        ↪ False):
ev = dev.wait_event([ "CTRL-EVENT-EAP-STARTED" ], timeout
    ↪ =10)
if ev is None:
    raise Exception(" Association_and_EAP_start_timed_out
        ↪ ")

```

```

ev = dev.wait_event([ "CTRL-EVENT-EAP-METHOD" ], timeout
    ↪ =10)

if ev is None:
    raise Exception("EAP_method_selection_timed_out")

if method not in ev:
    raise Exception("Unexpected_EAP_method")

if expect_failure:
    ev = dev.wait_event([ "CTRL-EVENT-EAP-FAILURE" ])
    if ev is None:
        raise Exception("EAP_failure_timed_out")
    ev = dev.wait_event([ "CTRL-EVENT-DISCONNECTED" ])
    if ev is None:
        raise Exception("Disconnection_timed_out")
    if not local_error_report:
        if "reason=23" not in ev:
            raise Exception("Proper_reason_code_for_
                ↪ disconnection_not_reported")

    return

ev = dev.wait_event([ "CTRL-EVENT-EAP-SUCCESS" ], timeout
    ↪ =10)

if ev is None:
    raise Exception("EAP_success_timed_out")

if initial:

```



```

        ev = dev.wait_event([ "CTRL-EVENT-CONNECTED" ],
                               ↪ timeout=10)
    else:
        ev = dev.wait_event([ "WPA: _Key_negotiation_completed
                               ↪ " ], timeout=10)
    if ev is None:
        raise Exception(" Association_with_the_AP_timed_out")
    status = dev.get_status()
    if status["wpa_state"] != "COMPLETED":
        raise Exception(" Connection_not_completed")

    if status["suppPortStatus"] != "Authorized":
        raise Exception(" Port_not_authorized")
    if method not in status["selectedMethod"]:
        raise Exception(" Incorrect_EAP_method_status")
    if sha256:
        e = "WPA2-EAP-SHA256"
    elif rsn:
        e = "WPA2/IEEE_802.1X/EAP"
    else:
        e = "WPA/IEEE_802.1X/EAP"
    if status["key_mgmt"] != e:
        raise Exception(" Unexpected_key_mgmt_status:_ " +
                               ↪ status["key_mgmt"])

```

```
def eap_reauth(dev, method, rsn=True, sha256=False):  
    dev.request("REAUTHENTICATE")  
    eap_check_auth(dev, method, False, rsn=rsn, sha256=  
        ↪ sha256)
```