

## University of New Hampshire University of New Hampshire Scholars' Repository

---

Master's Theses and Capstones

Student Scholarship

---

Spring 2007

# Simple authentication and security layer incorporating extensible authentication protocol

Myung-Sun Kim

*University of New Hampshire, Durham*

Follow this and additional works at: <https://scholars.unh.edu/thesis>

---

### Recommended Citation

Kim, Myung-Sun, "Simple authentication and security layer incorporating extensible authentication protocol" (2007). *Master's Theses and Capstones*. 270.

<https://scholars.unh.edu/thesis/270>

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact [nicole.hentz@unh.edu](mailto:nicole.hentz@unh.edu).

**SIMPLE AUTHENTICATION AND SECURITY LAYER INCORPORATING  
EXTENSIBLE AUTHENTICATION PROTOCOL**

BY

MYUNG-SUN KIM

B.S., University of New Hampshire, 2005

B.A., University of New Hampshire, 2005

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

In

Computer Science

May, 2007

UMI Number: 1443612

### INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

**UMI<sup>®</sup>**

---

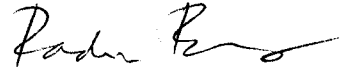
UMI Microform 1443612

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

This thesis has been examined and approved.



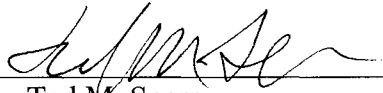
---

Thesis Director, Dr. Radim Bartoš  
Associate Professor of Computer Science



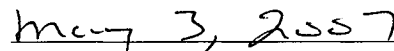
---

Scott A. Valcourt  
Research Project Manager of Computer Science



---

Dr. Ted M. Sparr  
Professor of Computer Science

  
Date

# DEDICATION

To my father, Young-Joo Kim

to my mother, Ae-Ja Kim

to my sister, Min-Sun Kim

to my brother-in-law, Jong-Seog Jung

to my nephew, Sae-Jun Jung

and to my wife, Sissy Kim.

# ACKNOWLEDGEMENT

I would like to express my appreciation to all those who made it possible for me to finish my master thesis here at UNH.

I would like to thank my advisor, Dr. Radim Bartoš, and my Research Project Manager, Mr. Scott A. Valcourt for their support, time, and helpful feedback during my master studies as well as my undergraduate studies. Their patience and their guidance throughout my whole master thesis gave me a great experience here at UNH. I also wish to thank my thesis committee, Dr. Ted M. Sparr, for his time and consideration.

Lastly, I would like to express my gratitude to the Meetinghouse Data Communication, Inc. (now part of Cisco Systems, Inc.) and the National Institute of Justice; Datacasting Project for Project54<sup>TM</sup>, for funding my graduate work.

# TABLE OF CONTENTS

DEDICATION.....	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF TABLES.....	vii
LIST OF FIGURES .....	viii
ABSTRACT.....	ix
<b>1 INTRODUCTION.....</b>	<b>1</b>
<b>2 BACKGROUND .....</b>	<b>5</b>
2.1 Terminology.....	5
2.2 Simple Authentication and Security Layer (SASL).....	7
2.3 Extensible Authentication Protocol (EAP) .....	9
2.4 EAP-Pre-Shared Key (EAP-PSK) .....	12
2.5 EAP-Transport Layer Security-PSK (EAP-TLS-PSK) .....	12
2.6 Motivations and Goals .....	13
<b>3 EXTENSIBLE AUTHENTICATION PROTOCOL – ADVANCED ENCIPHERMENT STANDARD – PRE-SHARED KEY (EAP-AES-PSK).....</b>	<b>15</b>
3.1 Approach.....	16
3.2 EAP-AES-PSK Message Flows and Packet Format.....	18
3.3 Features of EAP-AES-PSK.....	24
3.4 EAP-AES-PSK Supporting Aspects.....	25

3.4.1	Fragmentation .....	25
3.4.2	Channel Binding .....	26
3.4.3	Fast Reconnection .....	26
3.4.4	Key Strength .....	27
3.4.5	Number of Message Exchanges .....	28
3.4.6	Tunnel Support .....	28
3.5	Protections against Security Risks .....	28
3.5.1	Dictionary Attack .....	29
3.5.2	Replay Attack .....	30
3.5.3	Man-In-The-Middle (MITM) Attack .....	30
3.5.4	Re-Keying .....	32
3.5.5	Data Encryption .....	32
<b>4</b>	<b>VERIFICATIONS .....</b>	<b>33</b>
4.1	Outcomes .....	33
4.2	Evaluation .....	34
4.2.1	Verification of Functionality of a SASL Application incorporating EAP .....	34
4.2.2	Performance Evaluation .....	40
<b>5</b>	<b>CONCLUSION .....</b>	<b>47</b>
	<b>BIBLIOGRAPHY .....</b>	<b>49</b>
	<b>APPENDICES .....</b>	<b>51</b>
	<b>APPENDIX A ABBREVIATION AND ACRONYMS .....</b>	<b>52</b>
	<b>APPENDIX B EXPERIMENTS/USER GUIDES .....</b>	<b>53</b>
B.1	Cyrus Installation .....	53
B.2	Cyrus usage of ANONYMOUS mechanism with file transfer .....	54
B.3	Cyrus Password Setup (using sasldb) .....	54
B.4	Add Mechanism Plugins .....	55



# LIST OF TABLES

2-1	Terminology.....	5
3-1	Features of Pre-Shared Key Methods .....	24
4-1	Configuration of EAP-AES-PSK.....	35
4-2	95% Confidence Level Authentication Time for each Method .....	42
4-3	95% Confidence Level Completion Time in the Controlled Environment .....	46
4-4	95% Confidence Level Completion Time in the Un-Controlled Environment .....	46

# LIST OF FIGURES

1-1	SASL Incorporating EAP Abstract Layer.....	4
2-1	SASL Abstract Layer.....	7
2-2	Authentication Protocols .....	9
2-3	(a) EAP Message Exchange without AAA, (b) EAP Message Exchange .....	11
2-4	EAP Packet Format .....	12
3-1	Key Setup Algorithm in EAP-AES-PSK.....	17
3-2	Key Derive Algorithm in EAP-AES-PSK.....	17
3-3	EAP-AES-PSK Packet Format.....	19
3-4	EAP-AES-PSK Authentication Message Exchanges .....	20
3-5	Dictionary Attack.....	29
3-6	Man-In-The-Middle (MITM) Attack.....	31
4-1	Authentication Time for each Method.....	42
4-2	(a) Server Completion Time in the Controlled Environment, (b) Client Completion Time in the Controlled Environment.....	44
4-3	(a) Server Completion Time in the Un-Controlled Environment, (b) Client Completion Time in the Un-Controlled Environment.....	45

**ABSTRACT**

**SIMPLE AUTHENTICATION AND SECURITY LAYER**

**INCORPORATING EXTENSIBLE AUTHENTICATION PROTOCOL**

by

Myung-Sun Kim

University of New Hampshire, May, 2007

There are many methods that support user authentication and access control, important roles in the establishment of secure communication. Particularly, we examine Simple Authentication and Security Layer (SASL) and Extensible Authentication Protocol (EAP) and propose EAP-Advanced Encryption Standard-Pre-Shared-Key (EAP-AES-PSK). SASL is an authentication framework in connection-oriented protocols. EAP is an authentication framework providing multiple authentication methods. SASL is vulnerable to the dictionary attack, replay attack, and Man-In-The-Middle attack as well as the re-keying issue. We propose to incorporate EAP into SASL to enhance the security of SASL and to provide a pathway for easy incorporation of future EAP enhancements into SASL. Standalone EAP still faces some common attacks. We propose EAP-AES-PSK, a new EAP method, to provide strong authentication and we implement this method on the Cyrus SASL implementation: one of the publicly available SASL implementations. This project is evaluated through the verification of functionality of a SASL application incorporating EAP. Further, we argue how the common security risks associated with SASL are addressed, and we complete a performance evaluation of the new method incorporated into SASL.

# **CHAPTER 1**

## **INTRODUCTION**

Authentication and access control of users are important functions in the establishment of secure communication. Both senders and receivers should be able to verify the identity the other party. However, successful authentication does not always occur over insecure network connections. One of the commonly used authentication approaches is to authenticate via user identification (ID) and user password, or via host IP address. These methods are subject to the risks of insecure communication when an intruder gains physical access of the communication medium between users and hosts, such as what occurs in the dictionary attack, replay attack, and Man-In-The-Middle (MITM) attack. Therefore, software developers must consider network security when they design and develop network enabled software programs.

Simple Authentication and Secure Layer (SASL) [17] protocol provides a straightforward solution for the software developer without a deep understanding of network security. SASL provides an authentication framework in connection-oriented protocols and a secure data transportation mechanism between a server and a client that is detached from existing application protocols. The SASL framework allows either new applications to reuse existing authentication protocol mechanisms or allows old applications to use new authentication protocols without requiring any modifications of

the SASL library [17]. SASL has been incorporated into a few widely used protocols such as the Lightweight Directory Access Protocol (LDAP), Post Office Protocol 3 (POP3), the Internet Message Access Protocol (IMAP), and Telnet to enable pluggable authentication [23]. Currently, there are two publicly available SASL libraries: the Cyrus SASL library from Carnegie Mellon University and the GNU SASL library.

Even though SASL provides a standardized framework for an application supporting multiple authentication methods, the SASL authentication methods are vulnerable to many common attacks, such as dictionary attack, replay attack, and Man-In-The-Middle (MITM) attack, in addition to insecure data transportation. Therefore, we propose a method to protect against these attacks by integrating Extensible Authentication Protocol (EAP) [2] into SASL.

EAP is an authentication framework providing multiple authentication methods. Since EAP is not a specific authentication mechanism, the exact authentication type is chosen by negotiating between a peer and an authenticator. EAP runs over data link layers where the Internet Protocol (IP) is not required, e.g., Point-to-Point Protocol (PPP) [21] or Institute of Electrical and Electronics Engineers (IEEE) 802 standard protocols. Currently, there exist over 40 different EAP methods [2] to provide strong authentication services and secure data transmission.

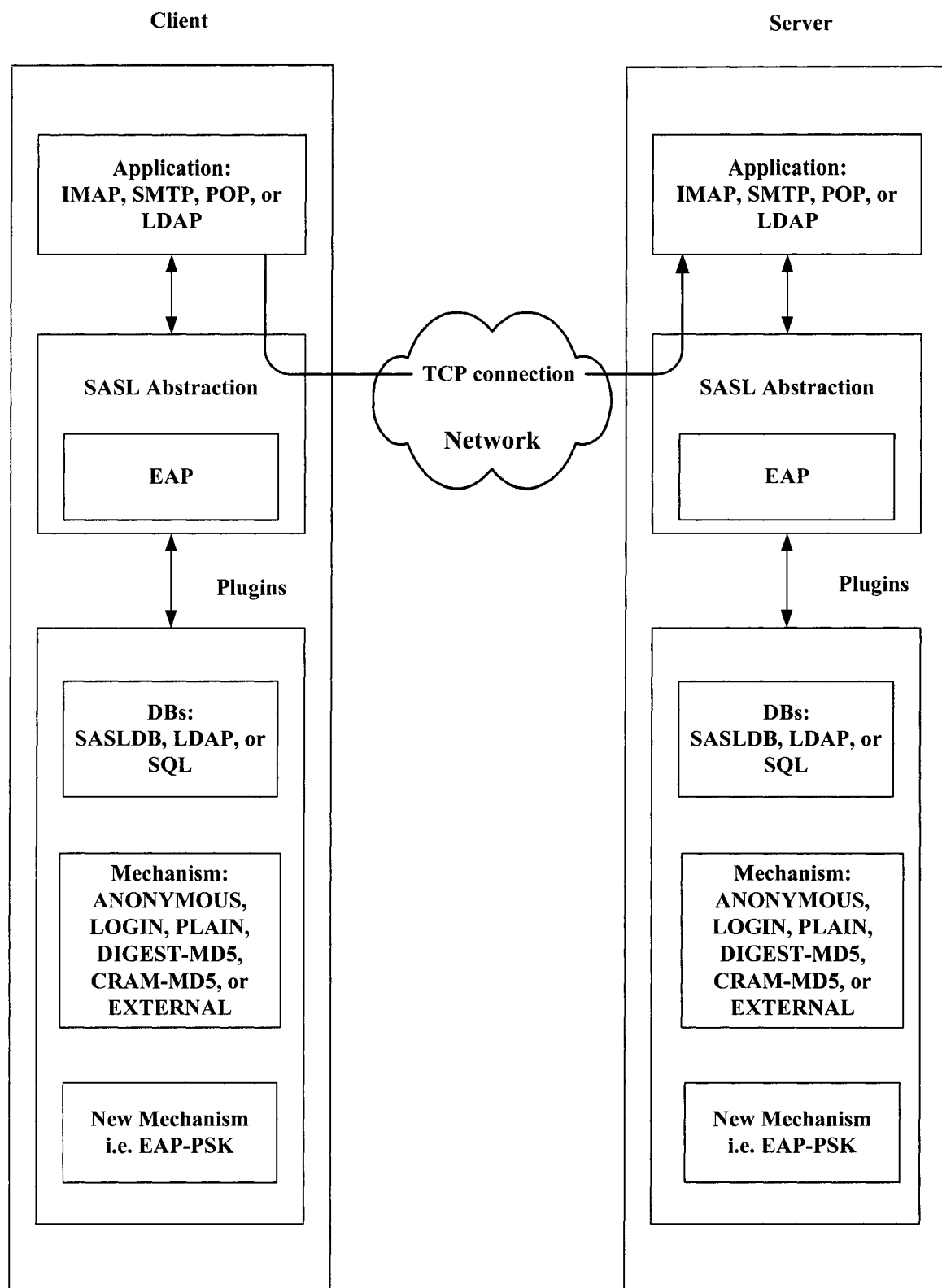
This document outlines the benefits and the problems in standalone SASL, followed by the advantages we will gain by combining EAP and SASL. Further, we present details on how to integrate EAP into SASL, so that SASL incorporating EAP addresses the existing weaknesses in SASL. This work builds upon a new EAP method, EAP - Advanced Encryption Standard - Pre-Shared Key (EAP-AES-PSK) that is an

updated version of EAP - Transport Layer Security-Pre - Shared-Key (EAP-TLS-PSK) [14].

Cyrus SASL implementation, randomly chosen for this project, incorporates several types of plugins: mechanism plugins, user canonization plugins, and auxiliary property plugins. Mechanism plugins are for authentication methods. User canonization plugins are used to differ the ways of canonizing authentication and authorization IDs. Auxiliary property plugins enable auxiliary properties about user credentials to be accessible, such as Authentication, Authorization and Accounting (AAA) services [18].

This project proposes to incorporate EAP into SASL to enhance security of SASL and to provide a pathway for easy incorporation of future EAP enhancements into SASL as well as to update and to implement EAP-AES-PSK. SASL incorporating EAP typically works by placing the EAP interface into the SASL library. The EAP interface not only bridges between the SASL library and existing SASL authentication methods, but also links the SASL library and future EAP authentication methods.

Figure 1-1 illustrates the architecture of SASL incorporating EAP including where the EAP interface is located in the SASL library directory and how the EAP interface connects SASL with legacy and future authentication methods. A more detailed explanation is presented in Chapter 2.



**Figure 1-1: SASL Incorporating EAP Abstract Layer**

## CHAPTER 2

# BACKGROUND

We present background details of Simple Authentication and Security Layer (SASL), Extensible Authentication Protocol (EAP), EAP-Pre-Shared Key (EAP-PSK), and EAP-Transport Layer Security-PSK (EAP-TLS-PSK).

### 2.1 Terminology

This document frequently uses the following terms:

**Table 2-1:** Terminology

<b>AAA server</b>	The entity that provides Authentication, Authorization, and Accounting services to an authenticator. In this document, the terms "AAA server" and "authentication server" are used interchangeably. The authentication server is typically a Remote Authentication Dial-In User Service (RADIUS) server.
<b>Authentication Key (AK)</b>	A static long-lived key, derived from the Pre-Shared Key (PSK), is to mutually authenticate the EAP peer and the EAP server [2].
<b>Authentication</b>	The process to verify the user identity in a communication such as requiring to log in.
<b>Client</b>	The end of the link that responds to the authenticator. This end is known as the supplicant in [12] and the peer in [2]. In this document, the terms "peer", "supplicant", and "client" are used interchangeably.



**Table 2-1** Continued

<b>Extended Master Session Key (EMSK)</b>	Additional keying material derived between the EAP client and server that is exported by the EAP method. The EMSK is at least 64 octets in length. The EMSK is not shared with the authenticator or any other third party. The EMSK is reserved for future uses that are not yet defined.
<b>Key Derivation Key (KDK)</b>	A static long-lived key, derived from the Pre-Shared Key (PSK), is to derive session keys [2].
<b>Master Session Key (MSK)</b>	Master Keying material that is derived between the EAP peer and server and exported by the EAP method. The MSK is at least 64 octets in length. In existing implementations, an AAA server acting as an EAP server transports the MSK to the authenticator.
<b>Nonce</b>	A number used once, such as a random number.
<b>Pre-Shared Key (PSK)</b>	A key in symmetric cryptography that is installed/sent to the opposite side of the connection in advance, or that is distributed by third party.
<b>Server</b>	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 where the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.
<b>Successful Authentication</b>	In this document, "successful authentication" is involved both authentication and authorization; even though the peer may pass authentication to the server, the peer may not have access to the server due to policy reasons [2]. For instance, the peer has a basic services right on a server and the peer tries to access a premium services on the server. The peer successfully authenticates to the authenticator, but is not able to access to the server.

## 2.2 Simple Authentication and Security Layer (SASL)

SASL is a framework to incorporate authentication in connection-oriented protocols and secure data transportation between a server and client that is detached from application protocols.

The SASL framework allows either new applications to reuse existing authentication protocol mechanisms, or allows old applications to use new authentication protocols without requiring any modifications to the SASL library [17].

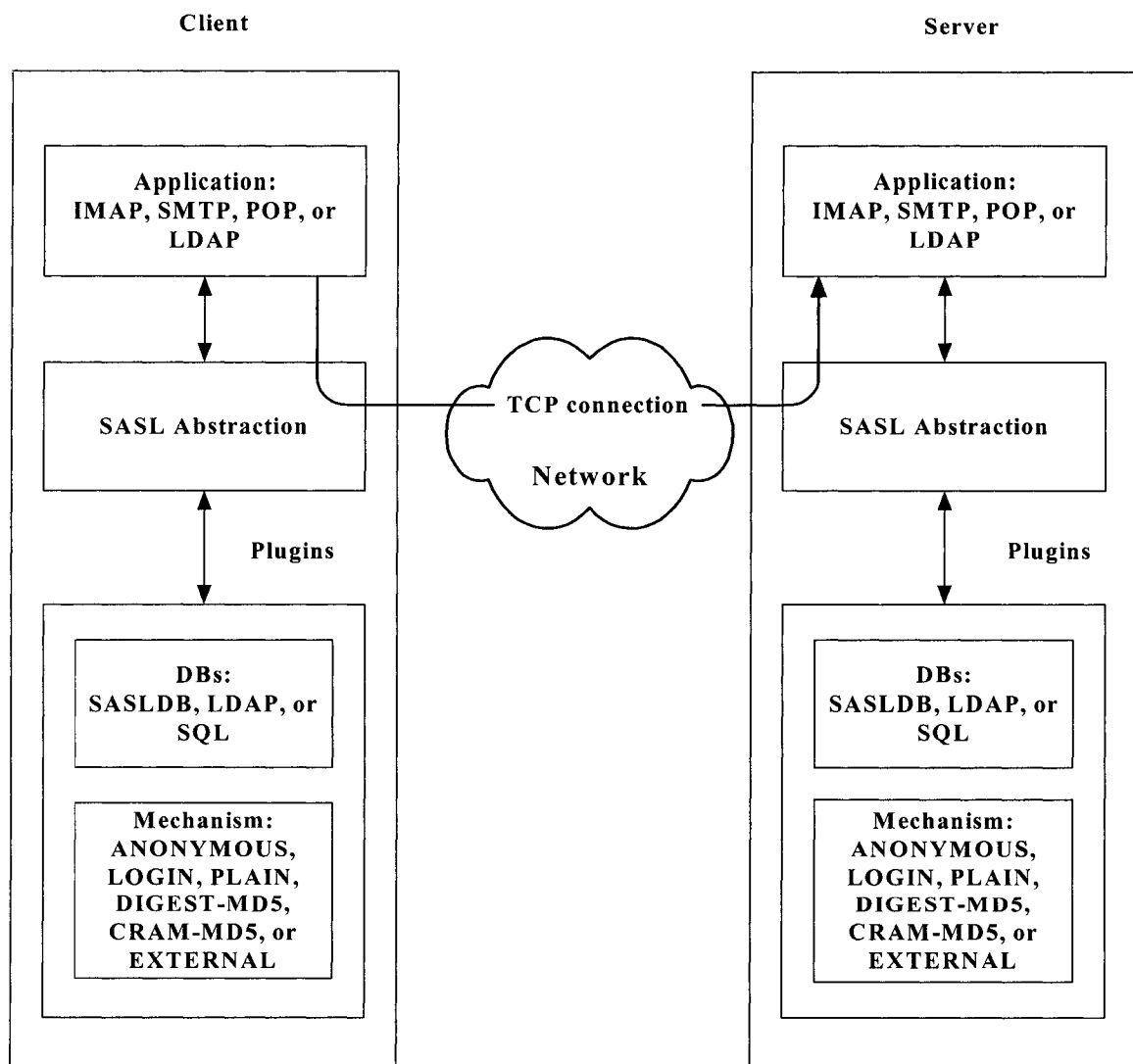


Figure 2-1: SASL Abstract Layer

Figure 2-1 illustrates the SASL abstract layer. Both client and server sides in the figure have three parts: an application, the SASL abstraction, and various plugins. The application is generally some connection-oriented protocol, such as Internet Message Access Protocol (IMAP), Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP), or Lightweight Directory Access Protocol (LDAP). Plugins contain database system and authentication protocol mechanisms. The SASL abstraction in the client and the server communicate to authenticate and authorize the connection, while the SASL abstraction monitors its application and plugins.

Both client and server sides in Figure 2-1 are symmetric; however, it is not required. For example, in the client side, database system SASL is not necessary because only the server side maintains user credential information within database systems. The client does not need to have all of the authentication methods that the server has available or vice-versa. When a client has only ANONYMOUS SASL method [24] that does not require authentication for guest access, SASL works fine if a server provides the ANONYMOUS SASL method with other authentication methods. However, if the server does not support the ANONYMOUS SASL method, the server will not authenticate the client. In order to access a server, a client must have one of the authentication methods supported by the server.

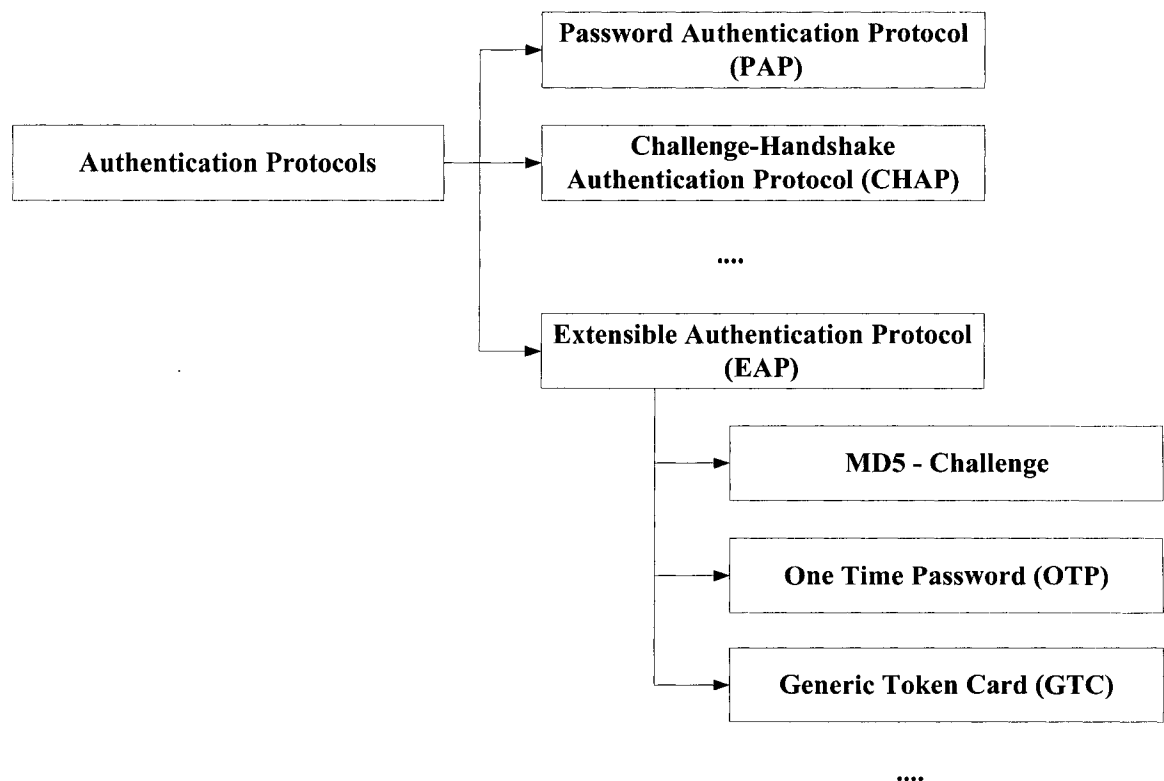
GNU and Cyrus SASL libraries support the applications: LDAP, IMAP, SMTP, and POP, and provide several authentication methods such as ANONYMOUS (for unauthenticated guest access), PLAIN (a simple clear text password mechanism), LOGIN (a simple password mechanism), CRAM-MD5 (a simple challenge-response scheme

based on HMAC-MD5 [7]), and EXTERNAL (for where authentication is implicit in the context).

Although SASL has many advantages, standalone SASL is vulnerable to many common attacks and insecure data transportation, in addition to re-keying issues, because many existing SASL mechanisms are vulnerable to these attacks and do not provide secure data encryption. Many of these attacks are addressed further in this chapter.

## 2.3 Extensible Authentication Protocol (EAP)

EAP was originally developed for the extensible usage of authentication protocols defined in Point-to-Point Protocol (PPP) [21], such as Password Authentication Protocol (PAP) [RFC 1334] and Challenge Handshake Authentication Protocol (CHAP) [22]. Figure 2-2 describes authentication protocols hierarchy [4].



**Figure 2-2:** Authentication Protocols

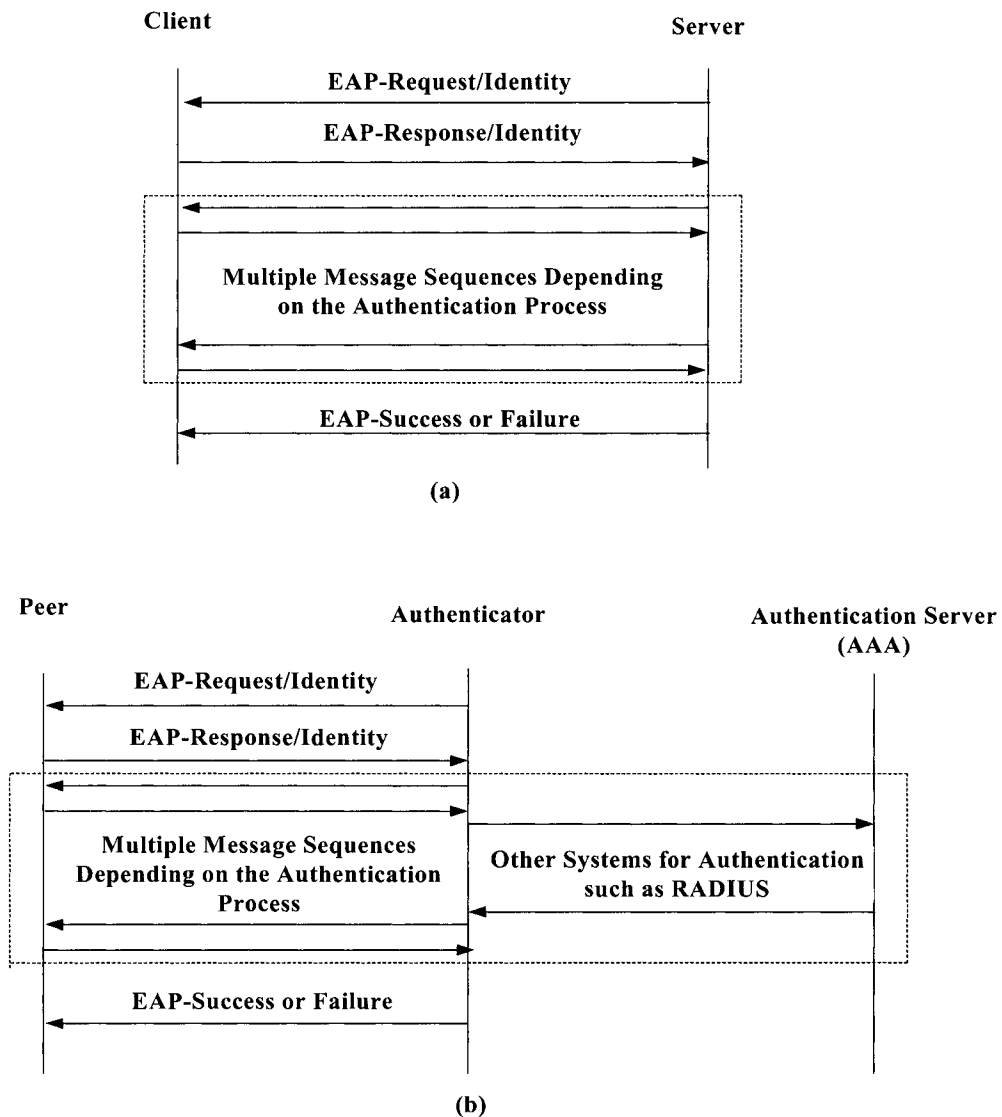
EAP is an authentication framework providing multiple authentication methods. Since EAP is not a specific authentication mechanism, the exact authentication type is chosen by negotiation between a peer and an authenticator. EAP provides its own support for avoiding duplication and retransmission, but not fragmentation. However, standalone EAP is not secure enough to authenticate and has various vulnerabilities [2].

Since all traffic within EAP methods is exposed to attackers, user identities are visible. EAP is vulnerable to replaying packets, dictionary attack, disrupting negotiation, as well as other vulnerabilities described in [2]. Even though standalone EAP methods are not acceptable over wireless networks or Virtual Private Networks (VPNs) due to those attacks, EAP is widely used in wired networks and Point-to-Point connections because EAP is an authentication framework, not a specific authentication mechanism. EAP provides a negotiation of the desired authentication mechanism [9]. A stronger authentication mechanism protects against the collection of vulnerable attacks while standalone EAP does not.

Figure 2-3 demonstrates an EAP message exchange. In EAP message exchanges, an EAP server requests user identity information from an EAP client and the client responds with the requested information back to the server. After the server obtains user identity information, more messages will be exchanged depending on the authentication process. The EAP server finally sends a message indicating whether the user authentication succeeds or fails.

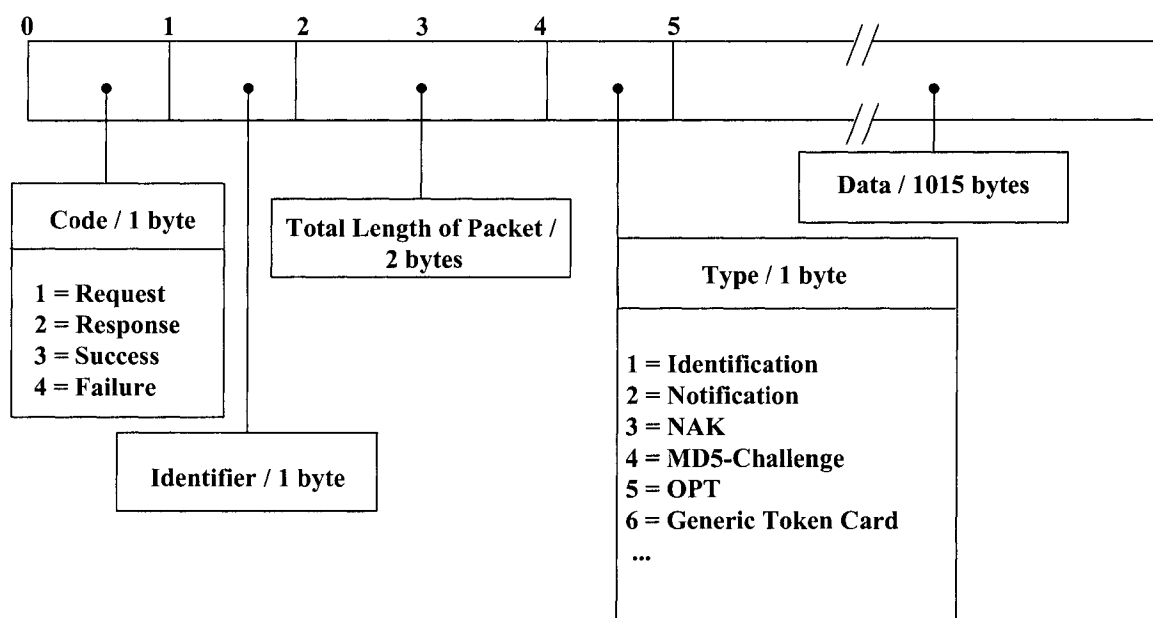
In Figure 2-3 (b) including Authentication, Authorization, and Accounting (AAA), the authentication server provides AAA services designed for ease in tracking usage for billing and network statistics purposes. Authentication server clients transfer user

credentials to the authentication server and the authentication server authenticates the user based on the provided credentials. The authentication server can be viewed as a kind of database and controller for usage billing and for network traffic management. However, this project mainly focuses on SASL and EAP, not on AAA services that the authentication server provides. Figure 2-3 (a) illustrates the EAP message exchange without AAA.



**Figure 2-3:** (a) EAP Message Exchange without AAA, (b) EAP Message Exchange

Figure 2-4 describes an EAP packet format.



**Figure 2-4: EAP Packet Format [2]**

## 2.4 Extensible Authentication Protocol – Pre-Shared Key (EAP-PSK)

A Pre-Shared Key EAP method (EAP-PSK) is one of the EAP methods that use PSK for mutual authentication and session key derivation. While EAP is a framework to authenticate, EAP-PSK is an actual authentication mechanism that utilizes PSK upon the EAP framework. EAP-PSK normally exchanges four messages in the authentication process between a client and a server. Once the authentication is successful, EAP-PSK provides a protected communication channel [5].

## 2.5 Extensible Authentication Protocol – Transport Layer Security – Pre-Shared Key (EAP-TLS-PSK)

EAP-TLS-PSK [14] is a new authentication method, EAP-PSK with TLS [6] tunneling support, as an EAP protocol that extends EAP-PSK, one of the authentication protocols based on pre-shared keys. EAP-TLS-PSK uses secure tunnel establishment by

TLS handshake to exchange authentication-related information between a client and a server, while EAP-PSK simply uses pre-shared keys to mutually authenticate. Authentication not based on certificates allows a reduced cost of management with fewer message exchanges between client and server.

EAP-TLS-PSK [14] consists of two primary phases: Transport Layer Security (TLS) Handshake Phase and TLS Tunnel Phase. The TLS Handshake Phase is similar to the EAP-TLS [8] handshake Phase. Rather than using the EAP server to generate a required certificate, the process uses the EAP-PSK scheme to authenticate mutually and to establish the secure protected channel necessary for further connectivity.

In the TLS Tunnel Phase, further authentication takes place in the initially-established secure tunnel from the TLS Handshake Phase. EAP-PSK with tunneling employs the EAP-Type Length Value (EAP-TLV) method [11], allowing richer communication between client and server, to further authenticate EAP identity. At the conclusion of the TLS Tunnel Phase, MSK is derived using the TLS Pseudo Random Function (PRF) [8] from inner authentications that are mixed with the inner secret Key-Derivation Key (KDK) [5] from the TLS Handshake Phase and the TLS client and server random numbers to protect subsequent data.

## **2.6 Motivations and Goals**

This project merges two technologies, SASL and EAP, to make the network more secure and introduces EAP-AES-PSK to provide strong authentication that is a new EAP method and will be built on SASL.



SASL provides key security services to a number of application protocols such as e-mail protocols, so that software engineers can have a handy security tool. Software developers focus on only their application level, not communication security issues. Currently publicly available SASL libraries are the Cyrus SASL library by Carnegie Mellon University and the GNU SASL library. Both SASL libraries are used in IMAP, LDAP, POP, and SMTP to provide secure authentication.

EAP is widely being deployed in environments such as wireless networks and used over the Internet to support multiple authentication methods, such as token cards, one-time passwords, Kerberos (a network authentication protocol using secret-key cryptography) [13], and certificates. EAP-AES-PSK, described in Chapter 3, is a new strong authentication method that will be built on the SASL implementation.

SASL incorporating EAP typically works by locating the EAP interface in SASL plugins. The EAP interface not only bridges between the SASL library and existing SASL authentication methods, but also links the SASL library and future EAP authentication methods, still allowing future authentication methods to use existing SASL mechanisms if necessary.

We recall that Figure 1-1 illustrates the architecture of SASL incorporating EAP including where the EAP interface is located in the SASL library directory and how the EAP interface connects SASL with legacy and future EAP authentication methods. This project proposes to incorporate EAP into SASL to enhance security of SASL and to provide a pathway for easy incorporation of future EAP enhancements into SASL.

In next chapter we propose an updated EAP-TLS-PSK which is called EAP - Advanced Encryption Standard - Pre-Shared-Key (EAP-AES-PSK).

## CHAPTER 3

# **EXTENSIBLE AUTHENTICATION PROTOCOL – ADVANCED ENCRYPTION STANDARD – PRE-SHARED KEY (EAP-AES-PSK)**

We propose a new EAP method: EAP - Advanced Encryption Standard - Pre-Shared Key (EAP-AES-PSK) that is modified from EAP - Transport Layer Security - PSK (EAP-TLS-PSK) for two primary reasons: to reduce the number of message exchanges between the authenticator and the peer, and to ease the implementation and management of Simple Authentication and Security Layer (SASL) while maintaining the same security level that EAP-TLS-PSK provides.

TLS is a cryptographic protocol that provides many security algorithms. A peer negotiates for an algorithm that TLS supports. However, TLS specifies that no data must be sent when TLS initiates. Therefore, if we want to continue to use TLS, we must modify the TLS implementation to be able to carry data in the initialization. We decided not to use TLS in order to ease the implementation and management, and the AES used in EAP-AES-PSK is one of the symmetric ciphers that TLS provides. As a benefit, there is no notoriously known attack in AES.

At first glance, it may appear that EAP-AES-PSK is the same as EAP-PSK. EAP-AES-PSK is similar to EAP-PSK, but is not the same. EAP-AES-PSK intends to authenticate twice (in Phase 1 and Phase 2) to allow the use of password-based legacy EAP methods in Phase 2 of EAP-AES-PSK, as well as to provide more security. At the end of each phase, the end points generate an outer Master Session Key (MSK) and an inner MSK. From the outer and inner MSKs, EAP-AES-PSK derives a MSK to protect subsequent data. EAP-AES-PSK supports fragmentation and reassembly.

This chapter explains the details of EAP-AES-PSK and discusses the possible attacks and insecure data exchanges in the currently available Simple Authentication and Security Layer (SASL) mechanisms, which lead to the motivation for the study of this area.

### **3.1 Approach**

EAP-AES-PSK, similar to other current EAP outer methods, consists of two primary phases. Phase 1 is similar to the EAP-PSK method. Rather than using the EAP server to generate a required certificate, the process uses the EAP-PSK scheme to authenticate mutually, generating a Transient EAP Key (TEK), to establish the secure protected channel necessary for further connectivity. At the beginning of Phase 1, a server and a client derive the Authentication Key (AK) and the Key Derivation Key (KDK) from PSK [5] such as following pseudo code in Figure 3-1. Note that actual implementation should be more efficient since this pseudo code is to provide straightforward explanation.

```

key_setup(psk, ...){
    coutner = 0
    set ak[16] with 0s
    set kdk[16] with 0s
    aes_128(psk, ak, ...)
    aes_128(psk, kdk, ...)
    ak ^= counter++
    kdk ^= counter
    aes_128(psk, ak, ...)
    aes_128(psk, kdk, ...)
}

```

**Figure 3-1: Key Setup Algorithm in EAP-AES-PSK [5]**

Both AK and KDK are 16 bytes. AK is used to mutually authenticate the EAP peer and the EAP server and KDK is used to derive other necessary keys. At the end of Phase 1, the outer master session key (MSK) derived from the KDK, peer nonce, and more such as `outerMSK = key_derive (KDK, RAND_P, ...)` in Figure 3-2. The outer MSK will be used as one of these inputs for deriving MSK after a successful authentication is made.

All MSKs of EAP-AES-PSK must be derived in the algorithm in Figure 3-2.

```

key_derive (kdk, rand_p, ...){
    hash[16]
    counter = 1
    aes_128(kdk, rand_p, hash, ...)
    hash ^= counter
    aes_128(kdk, hash, ...)
    hash ^= counter++
    for i = 0 to 3
        hash ^= counter
        aes_128(kdk, hash, msk[i * 16], ...)
        hash ^= counter++
}

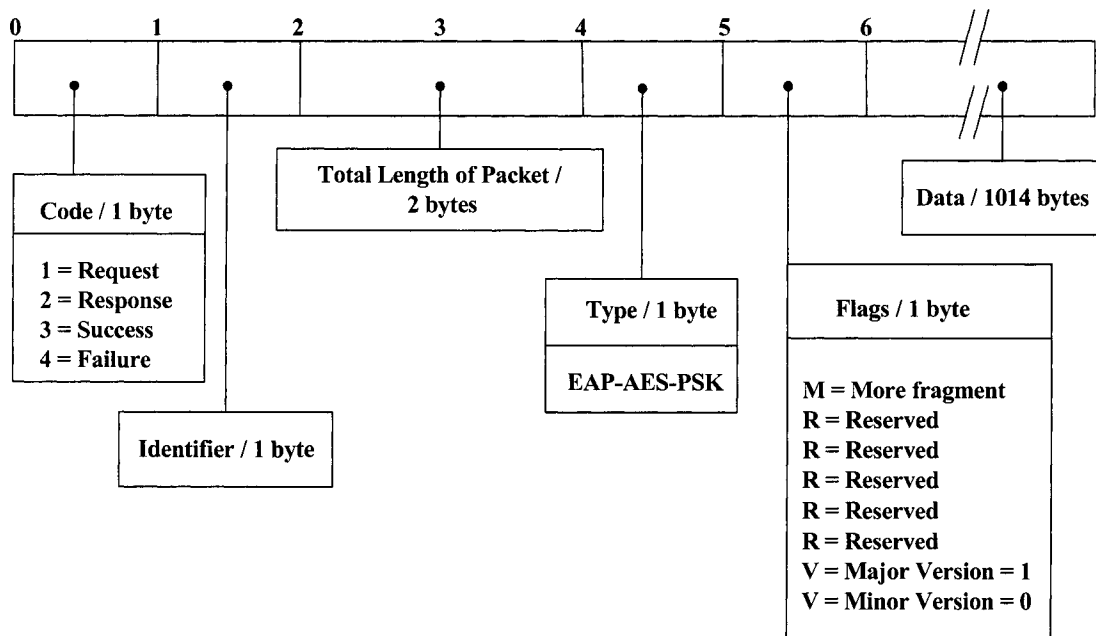
```

**Figure 3-2: Key Derive Algorithm in EAP-AES-PSK [5]**

Phase 2 performs further authentication in the initially-established AES secure tunnel from Phase 1. Unlike EAP-TLS-PSK, EAP-AES-PSK does not employ the EAP-TLV method to further authenticate EAP identity. At the conclusion of Phase 2, inner MSK is derived from KDK, peer secret (i.e., password), and more such `innerMSK = key_derive (KDK, password, ...)`. After successful authentication is made, MSK is derived from KDK, the concatenation of the first 8 bytes of outer and inner MSKs, and more such as `MSK = key_derive (KDK, outerMSK[0..7] || innerMSK[0..7], ...)`. MSK is to protect subsequent data. For more secure communication, MSK is needed to update in timely manner. For instance, every round trip both the server and the client renew the MSK to protect subsequent data.

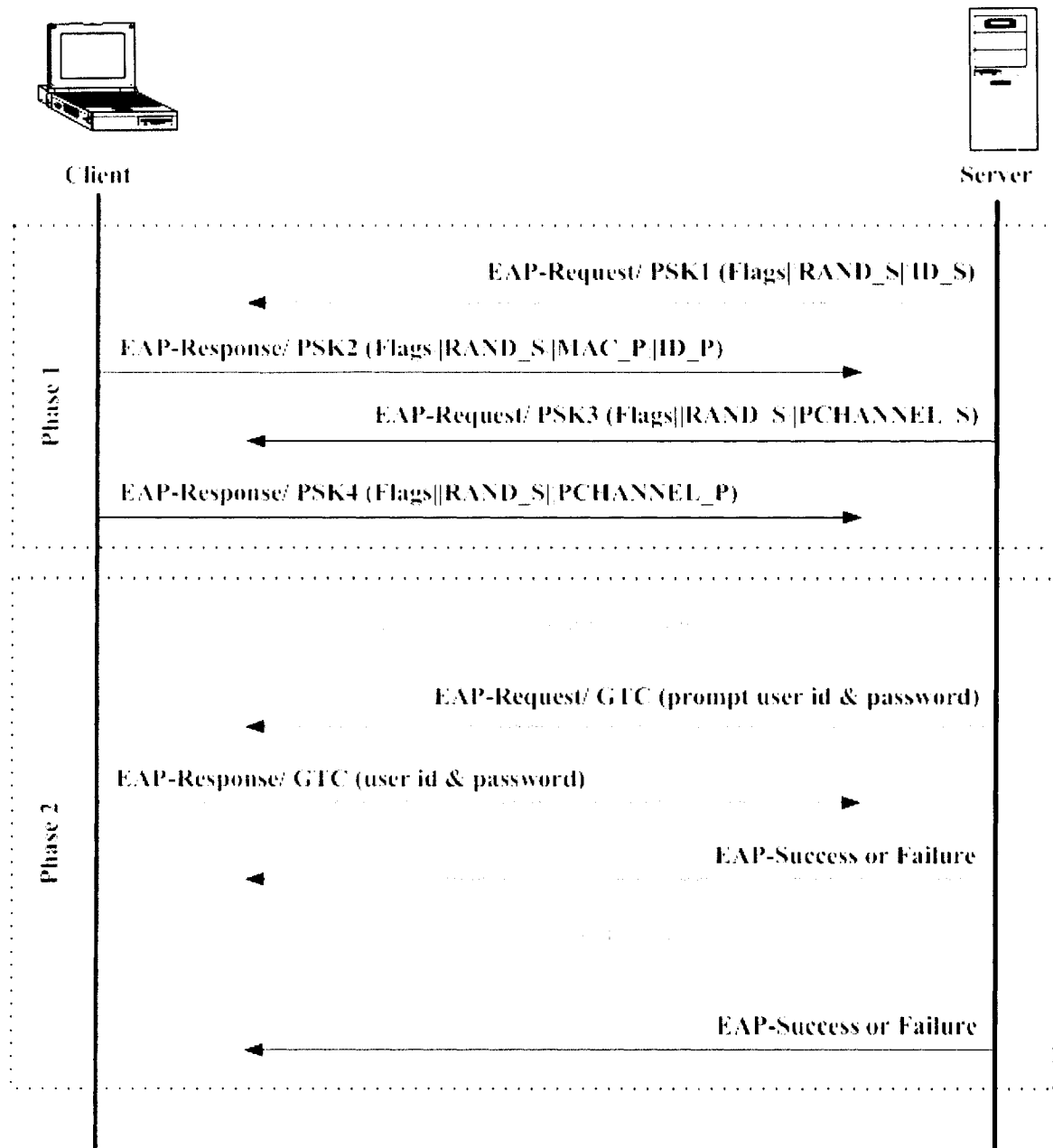
### **3.2 EAP-AES-PSK Message Flows and Packet Format**

The typical packet format of the EAP-AES-PSK messages would have the appearance as is described in Figure 3-3. The first byte of the EAP-AES-PSK packet indicates Request, Response, Success, or Failure. The second byte indicates message identifier for each round trip between a server and a peer. After each round trip, the identifier should be updated to the next available identifier from 0 to 255. The next two bytes contain the total length of the EAP-AES-PSK packet. The next byte is for the EAP type field. EAP-AES-PSK should set this value to 255 as this is an experimental EAP method. The next byte is for flags. The subsequent bytes are for data. Thus, EAP-AES-PSK needs at least 6 bytes for the header.



**Figure 3-3: EAP-AES-PSK Packet Format**

EAP-AES-PSK, as described in Section 3.1, may consist of two primary phases of message flows. EAP-AES-PSK standard authentication is comprised of four messages, i.e., two round trips; see Figure 3-4. EAP-AES-PSK assumes that the pre-shared key (PSK) is already known only to the EAP peer and EAP server and that the distribution of the PSK is outside of the scope of the EAP-AES-PSK mechanism. Note that ‘||’ within Figure 3-4 means concatenation.



**Figure 3-4: EAP-AES-PSK Authentication Message Exchanges**

In the EAP-AES-PSK method, the first message sent by the server to the peer contains:

- An EAP-Request with EAP-Type=EAP-AES-PSK;
- Flags : MRRRRRVV;

- M = More fragments
  - R = Reserved (must be zero)
  - R = Reserved (must be zero)
  - R = Reserved (must be zero)
  - R = Reserved (must be zero)
  - R = Reserved (must be zero)
  - V = 1 (major version)
  - V = 0 (minor version)
- The stated server identity (ID\_S); and
  - A server 16-byte random challenge (RAND\_S).

The second message is sent by the peer to the server containing:

- An EAP-Response with EAP-Type=EAP-AES-PSK;
- Flags: MRRRRRVV (major version = 1);
- A notice that it will authenticate to the server by proving that it is able to compute a particular Message Authentication Code (MAC\_P), which is a function of the two challenges and AK:

$$\text{MAC\_P} = \text{CMAC-AES-128}(\text{AK}, \text{ID\_P} || \text{ID\_S} || \text{RAND\_S} || \text{RAND\_P});$$

- A peer 16-byte random challenge (RAND\_P); and
- The stated peer identity (ID\_P).

Like EAP-PSK [5], Authentication Key (AK) is used to mutually authenticate the EAP peer and the EAP server. AK is a static long-lived key derived from the PSK. However, AK is not a session key.

The third message is sent by the server to the peer containing:

- An EAP-Request packet with EAP-Type=EAP-AES-PSK;
- Flags: MRRRRRVV (major version = 1);



- A notice that it will authenticate to the peer by proving that it is able to compute another MAC (MAC\_S), which is a function of the peer's challenge and AK:  $MAC\_S = CMAC-AES-128(AK, ID\_S || RAND\_P)$ ; and
- An established protected channel (PCHANNEL\_S) to:
  - Confirm that it has derived session keys (at least the TEK); and
  - Give a protected result indication of the authentication.

PCHANNEL\_S is an encrypted message by AES encryption with TEK that contains information about Phase 1 authentication result. If the client is able to decrypt PCHANNEL\_S, the server verifies the client's identity since TEK is derived directly from PSK and never transported over the Internet.

The fourth message is sent by the peer to the server containing:

- An EAP-Response packet with EAP-Type=EAP-AES-PSK;
- Flags: MRRRRRVV (major version = 1); and
- A notice that the establishment of the protected channel (PCHANNEL\_P) is complete to:
  - Confirm that it has derived session keys (at least the TEK); and
  - Give a protected result indication of the authentication.

PCHANNEL\_P is an encrypted message by AES encryption with TEK that contains information about Phase1 authentication result. If the server is able to decrypt PCHANNEL\_P, the client verifies the server's identity since TEK is derived directly from PSK and never transported over the Internet.

The internal method (in Phase 2) can be any other EAP method, such as EAP-GTC, and will allow for devices to establish safe authentication using the EAP-AES-PSK method. For instance, the internal method is encrypted by AES 128 encryption.

Figure 3-4 shows messages 5 through 8 are engaged in the exchange of user credentials using EAP-Generic Token Card (EAP-GTC) [2] exchanges. EAP-GTC has a data field in the Request includes a displayable message and a length of the packet of EAP-GTC is determined by adding message length to EAP-GTC header length. A Response must reply to the Request. We assume message 5 through 6 contain typical EAP header in Figure 3-3 with an EAP-Response packet with EAP-Type=EAP-AES-PSK.

The fifth message is sent by the server to the peer containing:

- An EAP-Request packet with EAP-Type=EAP-GTC; and
- A displayable message that prompts user id and password.

The sixth message is sent by the peer to the server containing:

- An EAP-Response packet with EAP-Type=EAP-GTC; and
- A response to the Request.

Both the seventh and eighth messages are sent by the server to the peer containing:

- An EAP-Success or Failure with no data.

Even though the seventh and eighth messages are exactly the same packets, we still need to send the eighth message to satisfy the definition of successful authentication. Successful authentication is defined in this document as involving both authentication

and authorization; even though the peer may pass authentication to the server, the peer may not access the server due to policy reasons. Thus, we need to send the eighth message although the message is a repeat of the previous seventh message. For instance, a peer has a basic service right on a server and the peer tries to access a premium service on the server. The peer probably passes the authentication, but not the authorization. In the case, the server sends EAP-Success within seventh message and EAP-Failure within eighth message.

While the EAP-AES-PSK method appears to offer 8 message exchanges at the minimum, both the outer and inner EAP methods are established using PSK and AES. This speedier method is expected to minimize message exchange latency that is found in other methods, while maximizing the feature set of this method.

### 3.3 Features of EAP-AES-PSK

Table 3-1 summarizes the features that EAP-TLS-PSK and EAP-AES-PSK provides. Both EAP methods propose tunneling support within EAP-PSK. The last column shows that EAP-AES-PSK brings fewer message exchanges between the client and server as compared to the EAP-TLS-PSK method. Details of the features are shown in Section 3.4.

**Table 3-1: Features of Pre-Shared Key Methods**

	<b>EAP-TLS-PSK [14]</b>	<b>EAP-AES-PSK (proposed)</b>
<b>Fragmentation/ Reassembly</b>	<b>Yes</b> By use of TLV Flags 'More Fragment' bit	<b>Yes</b> EAP-AES-PSK flags contain 'More Fragment' bit
<b>Channel Binding</b>	<b>Yes</b> Inherited from EAP-PSK	<b>Yes</b> Inherited from EAP-PSK

**Table 3-1 Continued**

<b>Fast Reconnection</b>	<b>Yes</b>	<b>Yes</b>
<b>Key Strength</b>	<b>128 bit</b> Inherited from EAP-PSK	<b>128 bit</b> Inherited from EAP-PSK
<b>Minimum # of Messages Exchanges</b>	<b>11</b> Phase 1 (4) + Phase 2 (7)	<b>8</b> Phase 1 (4) + Phase 2 (4)
<b>Tunnel Support</b>	<b>Yes</b> TLS tunnel	<b>Yes</b> AES tunnel
<b>Difficulty of Implementation</b>	<b>Harder</b> than EAP-AES-PSK	<b>Easier</b> than EAP-TLS-PSK

### 3.4 EAP-AES-PSK Supporting Aspects

This section explains details of EAP-AES-PSK supporting aspects such as fragmentation, channel binding, fast reconnection, key strength, number of message exchanges, and tunnel support.

#### 3.4.1 Fragmentation

The ideal EAP method supports fragmentation and reassembly. As RFC3748 states, EAP methods should support fragmentation and reassembly if EAP packets can exceed the minimum Maximum Transmission Unit (MTU) of 1020 bytes [2]. Fragmentation in EAP-TLS-PSK can be done through the use of flags within the TLV in Phase 2 only.

EAP-AES-PSK has its own 1 byte flags field such as MRRRRRVV in Figure 3-3. If the M bit is 1 in the received EAP packet, there will be more data. If following the size of the data is larger than the minimum MTU, M in the flags is set to 1. However

fragmentation and reassembly may be needed only in Phase 2 of EAP-AES-PSK. Since Phase 1 of EAP-AES-PSK is defined to support fragmentation and reassembly in its definition, MTU sizes of EAP-AES-PSK packets will not exceed 1020 bytes when normally transmitted in a network.

### **3.4.2 Channel Binding**

The verification within an EAP method of integrity-protected channel properties is the process of channel binding. For instance, endpoint identifiers within an EAP method can be compared to stored values (in AAA service systems).

EAP-AES-PSK uses an EAX mode of operation, an authenticated-encryption with associated data mode of operation for block ciphers, to provide channel binding as stated in Section 2.2.3 of EAP-PSK [5]. We explained earlier how EAP-AES-PSK achieves channel binding in Section 3.2.

### **3.4.3 Fast Reconnection**

The ability to create a new or refreshed security association, in the case where a security association has been previously established, in a more efficient manner or in a smaller number of round-trips is the goal of fast reconnection. Fast reconnection is optional in EAP-AES-PSK since, by having fast reconnection, EAP-AES-PSK saves one message round trip.

The server and peer must minimally cache the server and peer nonces, but not master secret, in order to have fast reconnection within EAP-AES-PSK. The master secret will be regenerated from the pre-shared key, server and peer nonce, and AES ciphersuite. The peer attempts to resume a session by sending an encrypted message

along with peer nonce from a previous session. The message should be encrypted using Transient EAP Key (TEK) derived from the pre-shared key and the AES ciphersuite. If the server has a matched peer nonce and is able to decrypt the message, the server will respond by sending an encrypted message to inform the peer of the server's successful verification of the peer. Then, EAP-AES-PSK Authentication Phase 1 establishes a secure tunnel and the conversation continues on to Phase 2. An example of fast reconnection message exchanges is shown in Section 4.2 that presents verification of EAP-AES-PSK functionality.

#### **3.4.4 Key Strength**

Following RFC3748 directives, if the method derives keys, then the effective key strength must be estimated. This estimate is meant for potential users of the method to determine if the keys produced are strong enough for the intended application. Since the keys within EAP-AES-PSK are derived by AES 128 ciphersuite like the EAP-PSK method, EAP-AES-PSK has the same key strength as EAP-PSK, which is 128 bit. However, the key strength of EAP-AES-PSK can be upgraded to 196 or 256 bit by replacing AES algorithm function. No one can be able to tell how long any cryptographic algorithm including the AES algorithm will last secure regardless key strength. As following National Institute of Standards and Technology (NIST) [1], AES algorithm approximately has  $3.4 \times 10^{38}$  possible 128-bit keys,  $6.2 \times 10^{57}$  possible 192-bit keys, and  $1.1 \times 10^{77}$  possible 256-bit keys.

### **3.4.5 Number of Message Exchanges**

This indicates the number of message exchanges between a client and a server. EAP-AES-PSK has 8 message exchanges required in a normal configuration, while EAP-TLS-PSK needs 11 message exchanges to establish a proper secure connection. Both EAP-AES-PSK and EAP-TLS-PSK need 4 message exchanges in Phase 1, but EAP-AES-PSK requires only 4 message exchanges in Phase 2 while EAP-TLS-PSK requires 7 message exchanges. Since EAP-AES-PSK does not require using Type Length Value (TLV) for crypto-binding in Phase 2, EAP-AES-PSK has 3 less message exchanges than EAP-TLS-PSK has.

### **3.4.6 Tunnel Support**

Unlike EAP-TLS-PSK, EAP-AES-PSK uses Advanced Encryption Standard (AES) [1] instead of Transport Layer Security (TLS). AES published by the NIST, is known to provide strong encryption and authentication. EAP-AES-PSK supports strong encryption to prevent eavesdropping and mutual authentication and to guarantee that user credentials are transmitted over legitimate networks. In the EAP-AES-PSK method, the fifth to the seventh exchanged messages are protected by AES encryption tunnel support. Section 3.2 explains how this works and Section 4.2 illustrates an example of actual messages in communication.

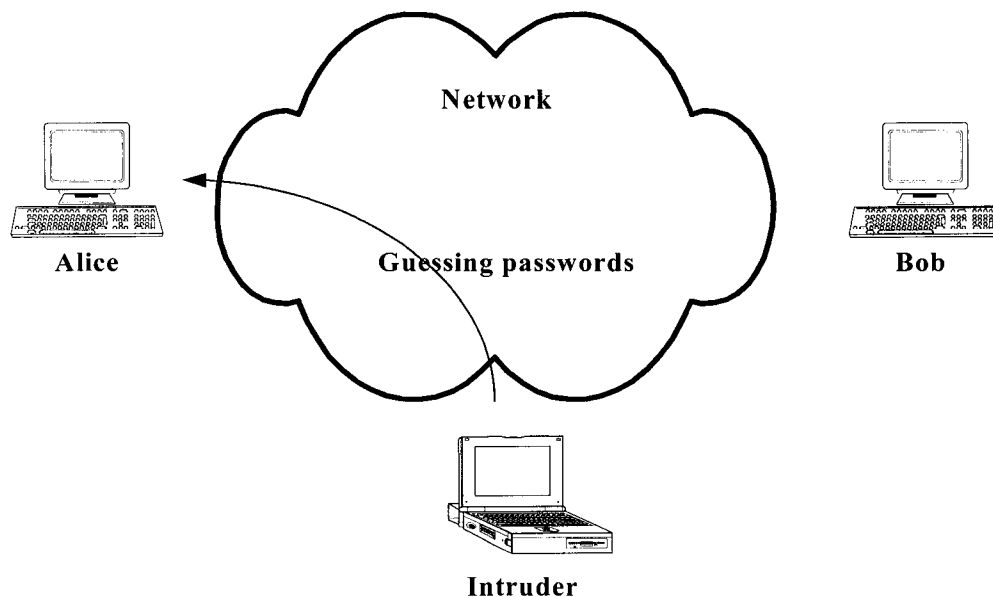
## **3.5 Protections Against Security Risks**

Typical attacks on Simple Authentication and Security Layer (SASL) mechanisms include the dictionary attack, replay attack, and Man-In-The-Middle (MITM) attack as well as re-keying issues. Extensible Authentication Protocols (EAP)

such as EAP-AES-PSK in SASL provides greater protection against the protections against those previously listed attacks based on the designs of both SASL and EAP and their combined usage.

### 3.5.1 Dictionary Attack

A dictionary attack is a technique used to get around an authentication mechanism for accessing a computer system by guessing typical passwords. In many cases, the credential exchanges are open to attacks, such as dictionary attacks, on a password. This attack could occur in SASL mechanisms that authenticate users by user ID and password. For instance, in Figure 3-5, the intruder guesses on Bob's password that matches with Bob's ID to gain access to Alice's system.



**Figure 3-5:** Dictionary Attack

EAP methods in SASL create a secured Advanced Encryption Standard (AES) tunnel between the client and the authentication server first. Some legacy SASL mechanisms, such as password based-authentication methods, could be used to



authenticate a client. Since user credential data transmission is performed only within a protected tunnel, SASL incorporating EAP methods is safe from the dictionary attack.

### **3.5.2 Replay Attack**

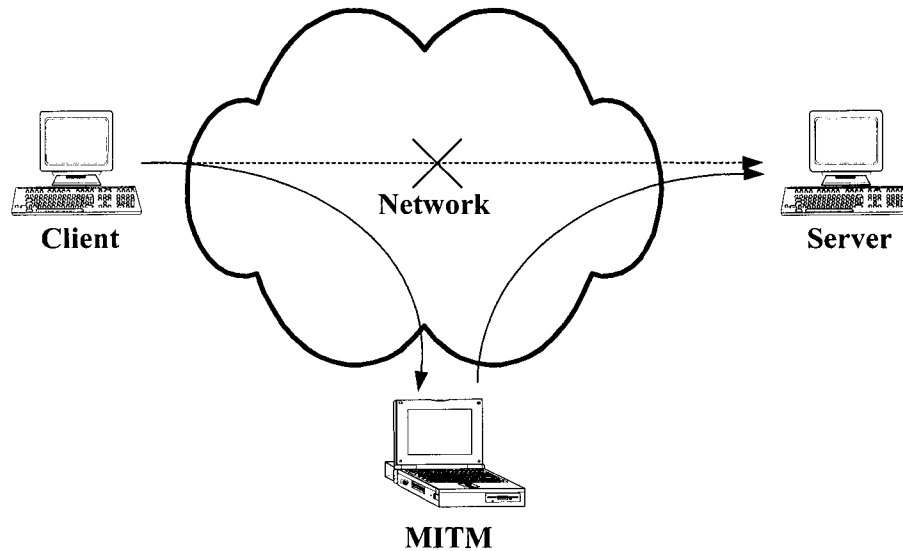
Attackers could obtain a message from a legitimate user with accesses into a network and replay that message later. This attack is called the replay attack. Replay attack causes unauthorized authentication or denial of service (DoS) [19].

The authentication process in EAP-AES-PSK is based not only on Pre-Shared key (PSK), but also on nonces such as server and client random numbers. EAP-AES-PSK server and clients exchange their nonces by including within the first and second messages of EAP-AES-PSK. Thus, the intruder will not gain successful authentication even if the intruder captures a message and resends to the server during the authentication process. The current server nonce will be not same as the server nonce from the last authentication process. Therefore, SASL incorporating EAP methods are safe from the replay attack.

### **3.5.3 Man-In-The-Middle (MITM) Attack**

Some mechanisms in SASL define authentication as a one-way action, such as a simple clear text password mechanism, PLAIN, although Generic Security Services Application Program Interface (GSSAPI [16]) [18], an application programming interface for programs to obtain security services [10], provide mutual authentication that uses Kerberos V. For example, when a client logs onto a network, only the server requires the client authentication. Because of one-way authentication, an attacker who resides between a client and a server can participate in the login exchange without the knowledge

of the client or the server (see Figure 3-6). This attack is known as the Man-In-The-Middle [3] attack.



**Figure 3-6:** Man-In-The-Middle (MITM) Attack

Typically MITM attack happens when mutual authentication is not performed. However, SASL incorporating EAP methods do mutual authentication between a client and a server. For instance, EAP-AES-PSK authenticates mutually via sending and receiving message 3 and 4 that contain PCHANNEL\_S and PCHANNEL\_P, encrypted messages by Advance Encryption Standard (AES) encryption. If the client is able to decrypt PCHANNEL\_S message from the server, the server achieves verifying the client's identity because TEK, a key to encrypt or decrypt a message, is derived directly from PSK and never transported over the Internet. If the server is able to decrypt PCHANNEL\_P, the client achieves verifying the client's identity. Thus, SASL incorporating EAP methods could protect against MITM attacks.

### **3.5.4 Re-keying**

The SASL framework does not provide re-keying services. Thus, cryptographic keys age and become insecure as they are used and as time goes by [9]. EAP generates keying materials, such as Master Session Key (MSK) and Extended Master Session Key (EMSK), during the authentication process. MSK, which is at least 64 octets in length, is a keying material that is derived between the client and the server and carried by the EAP method. The EMSK is extended from MSK and is reserved for future uses that are not yet defined [2]. Secure EAP methods such as EAP-AES-PSK renew MSK in time manner. For instance, EAP-AES-PSK updates keys every round trip between a server and a peer.

If SASL uses EAP keying services, future re-keying services of EAP will be automatically extended into the SASL mechanism and allow SASL incorporating EAP methods to support stronger cryptographic keys.

### **3.5.5 Data Encryption**

Existing SASL, as defined, does not strongly encrypt messages in a communication between a client and a server. Standalone SASL only provides a 64 bit encryption scheme. Thus, attackers could easily sniff messages in the communication by using network protocol analyzers.

EAP in SASL solves the data encryption issue. Data in EAP-AES-PSK is encrypted by AES 128 ciphersuite or higher. EAP generates MSK after a successful authentication thus the MSK is used in AES 128 ciphersuite to encrypt data. Therefore, SASL incorporating EAP provides secure data transfer.

## **CHAPTER 4**

# **VERIFICATIONS**

Currently, there are two publicly available Simple Authentication and Secure Layer (SASL) libraries available: Cyrus SASL library from Carnegie Mellon University and the GNU SASL library. In this project, Cyrus SASL library was selected randomly and used as part of the implementation as well as Hostapd and Wi-Fi Protected Access (WPA) supplicant [15] for SASL and EAP frames respectively. In Chapter 3 we proposed a new Extensible Authentication Protocol (EAP) method, EAP - Advanced Encryption Standard - Pre-Shared Key (EAP-AES-PSK). EAP-AES-PSK is successfully built on the SASL library and used for simple file transfer. This chapter details the project outcomes and evaluates the result.

### **4.1 Outcomes**

This project proposes to incorporate EAP into SASL to enhance the security of SASL and to provide a pathway for the easy incorporation of future EAP enhancements into SASL. The practical details on how to merge EAP into a SASL application source code library is stated in Appendix B, including documentation about SASL incorporating EAP such as implementation of SASL incorporating EAP: EAP interface, implementation of SASL incorporating EAP-PSK, implementation of SASL

incorporating EAP-AES-PSK, SASL Application: basic file transfer application requiring authentication, and a general SASL installation user guide, and a basic file transfer SASL application user guide.

## **4.2 Evaluation**

Evaluation of this project contains three parts: verification of functionality of a SASL application incorporating EAP, arguments to show how common security risks are addressed, and a performance evaluation. In Section 3.5, we already argue that SASL incorporating EAP will reduce or prevent the considerable security risks such as the dictionary attack, replay attack, and Man-In-The-Middle (MITM) attack as well as re-keying issues.

### **4.2.1 Verification of Functionality of a SASL Application incorporating EAP**

The outcomes specified in Section 4.1 above are performed for functionalities and secure data encryption by using network protocol analyzers. The network protocol analyzer examined transmissions to verify that the messages in the communication are secure. For instance, captured messages between a client and a server in a SASL application was compared with a result that we expect what the captured messages should be.

The configuration of EAP-AES-PSK must be preplanned before we use EAP-AES-PSK. Table 4-1 is a configuration example of EAP-AES-PSK. Both a server and a client must set the Pre-Shared Key and ID respectively to be “thisistemporarykey” and “EAP-AES-PSK\_server/ EAP-AES-PSK\_client.” The client needs to provide a user ID

and a user password for Phase 2 that will be used in the inner method, i.e., “user1” and “pass1.”

**Table 4-1: Configuration of EAP-AES-PSK**

	Server	Client
<b>Pre-Shared Key</b>	Thisistemporarykey	
<b>ID</b>	EAP-AES-PSK_server	EAP-AES-PSK_client
<b>User ID for Phase2</b>	N/A	user1
<b>User Password for Phase2</b>	N/A	pass1

The following boxed messages in octet form were captured during a SASL incorporating EAP-AES-PSK communicated. Each line starts with a 7 digit line number in octet form followed by 16 bytes, such as ‘od -x’ format, in UNIX-like machine format. Note that ‘||’ means concatenation and ‘n’ (in ‘[n]’) represents a number of byte(s), i.e., [3] means 3 bytes.

When EAP-AES-PSK is initialized, both the server and the client derive Authentication Key (AK) and Key Derivation Key (KDK) from the MSK above.

AK [16]:

```
00000000 eb69 f5eC 59dc 1cd8 2f11 cd36 6f29 3fd3
```

KDK [16]:

```
00000000 3c4b 340c e8e1 3780 9f58 f8be bbac 8d04
```

**First message (server → client):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_S [16] || ID\_S [18]):

```
0000000 0101 0028 ff02 2da5 f99e fc18 162b 2add
0000020 5534 7ba5 f88d 4541 502d 4145 532d 5053
0000040 4b5f 7365 7276 6572
```

EAP-Request, EAP-Response, EAP-Success, and EAP-Failure are defined to values 1, 2, 3, and 4, respectively.

**Second message (client → server):**

(EAP-Response [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_S [16] || RAND\_P [16] || MAC\_P [16] || ID\_P [18]):

```
0000000 0201 0048 ff02 2da5 f99e fc18 162b 2add
0000020 5534 7ba5 f88d 1a4c b33d 8f83 540d 1e96
0000040 077b ee3a 2dfa a0be dffe e543 821a cddd
0000060 ca9c 0c52 5a34 4541 502d 4145 532d 5053
0000080 4b5f 636c 6965 6e74
```

**Third message (server → client):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_S [16] || MAC\_S [16] || PCHANNEL\_S):

```
0000000 0102 003b ff02 2da5 f99e fc18 162b 2add
0000020 5534 7ba5 f88d 3306 7a2c 52fe 72df 33d0
0000040 e809 d83d e8da 0000 0000 c73d 04dc c807
0000060 1f9c 6e59 0e60 9cdc 13ae ed
```

**Fourth message (client → server):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_S [16] || PCHANNEL\_P):

```
0000000 0202 002b ff02 2da5 f99e fc18 162b 2add
0000020 5534 7ba5 f88d 0000 0001 4d59 d5c1 0362
0000040 0817 7c68 c053 4a2b e472 bb
```

**Fifth message (server → client):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
nonce [4] || tag [16] || EAP-Request [1] || identifier [1] || Length [2] || EAP-GTC [1] ||  
“UserID? Password?”):

```
0000000 0103 0032 ff02 0000 0000 2a4f ec4a eb73
0000020 7123 eaf3 9073 701e 04ae 6cbe c0a4 48fe
0000040 c8a3 8a2c 64d2 bb40 9cba dcb8 a680 7583
0000060 c86b
```

Nonce and tag are needed to establish the AES tunnel. Nonce should be 16 bytes.

The first 12 bytes are set to be zeros and only the last 4 bytes are to be defined. Thus,



from the second EAP-Request of the fifth message is where the inner EAP method starts encryption.

**Sixth message (client → server):**

(EAP-Response [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] || nonce [4] || tag [16] || EAP-Response [1] || identifier [1] || Length [2] || EAP-GTC [1] || “user1/pass1”):

```
0000000 0203 002b ff02 0000 0001 0a07 c1cc 67fb
0000020 b47c 000a 3c45 e500 b200 393e 82cd 2cd7
0000040 2dba 41d7 cd10 7327 b826 75
```

**Seventh message (server → client):**

(EAP-Response [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1]):

```
0000000 0304 0006 ff02
```

EAP-AES-PSK needs 8 message exchanges. However EAP-AES-PSK within the SASL framework does not require the eighth message, which is exactly the same as the seventh message. We may send the seventh message again for the last message, but it wastes time and is unnecessary because the SASL framework is required to send “SASL\_OK”, indicating successful authentication is made, and that can replace the eighth message of EAP-AES-PSK in the EAP framework.

After successful authentication in SASL incorporating EAP (using EAP-AES-PSK), the Master Session Key (MSK) is derived to protect subsequent data from manipulating the outer and inner MSKs using KDK.

Outer MSK [64]:

```
0000000 33d8 4480 e8da 644f e090 78c6 0a5d 97c6
0000020 534f b7a7 d899 6afd 62c8 593d 76e0 3635
0000040 8896 8f1f 9daa b463 6dbc 7ff0 f368 2d52
0000060 cddb e146 59fe e84d fedd 3fed 15e4 6d4d
```

Inner MSK [64]:

```
0000000 0440 ef76 6652 9b29 42a8 cbd0 4946 5891
0000020 8722 f109 02ef 5062 0b60 8859 4ade 46c5
0000040 2bdb 02a0 a87a d242 e6d8 3421 0c82 b43a
0000060 4def 7612 a6bd 0850 dee2 24ad b27D fb32
```

MSK [64]:

```
0000000 ca99 f0ec 8a13 c908 aff9 68f5 24cb 9eb1
0000020 440c d40a 77b0 3373 cf0e 22dd 8353 530c
0000040 7e92 9611 3543 88f8 f90b 2077 9193 0e77
0000060 5f87 6561 ccd1 481b ed58 cae5 e9a1 d31c
```

The following two extra messages are from fast reconnection, described in Section 3.4, after the successful authentication above. Both the server and the client saved their nonces.

**First message in fast reconnection (client → server):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_P [16] || PCHANNEL\_P):

```
0000000 0107 002b ff02 1a4c b33d 8f83 540d 1e96
0000020 077b ee3a 2dfa 0000 0000 c73d 04dc c807
0000040 1f9c 6e59 0e60 9cdc 13ae ed
```

**Second message in fast reconnection (server → client):**

(EAP-Request [1] || identifier [1] || Total Length [2] || EAP-AES-PSK [1] || flags [1] ||  
RAND\_S [16] || PCHANNEL\_S):

```
0000000 0207 002b ff02 2da5 f99e fc18 162b 2add
0000020 5534 7ba5 f88d 0000 0001 4d59 d5c1 0362
0000040 0817 7c68 c053 4a2b e472 bb
```

If the server and the client are able to decrypt the message that is encrypted by each other, then EAP-AES-PSK Authentication Phase 1 establishes a secure tunnel and the conversation continues on to Phase 2. Otherwise, both the server and the client start normal authentication.

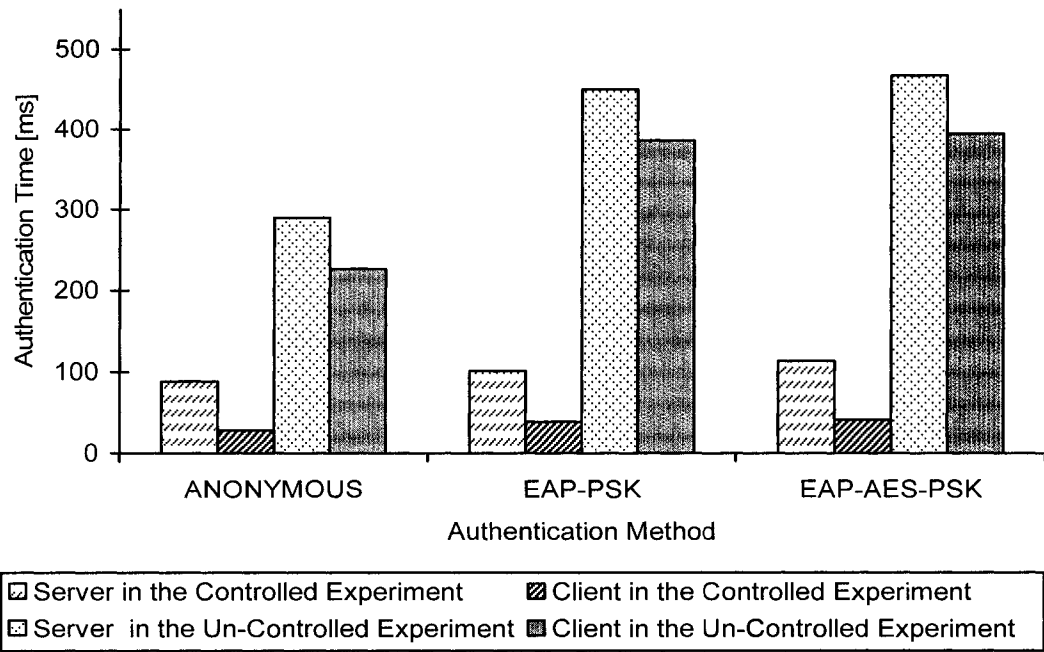
#### **4.2.2 Performance Evaluation**

**Authentication Time Test (Access Time Test):**

The authentication time test was executed on the SASL only-application (i.e., ANONYMOUS), SASL using EAP-PSK method, and SASL EAP-AES-PSK. Figure 4-1 and Table 4-2 show the amount of time needed to perform authentication in ANONYMOUS, EAP-PSK, and EAP-AES-PSK. Note that all performance tests were executed on machines with 500 MHz CPU/256 MB memory and 700 MHz CPU/256 MB memory for the server side and the client side respectively.

The server is always running and waiting for a client connection. Thus, server authentication time is measured from the time a new client connects to the time the client disconnects. Client authentication time is measured from the time the client initializes to the time the client ends.

The authentication time for the each method are measured in both the controlled environment and the un-controlled environment. Figure 4-1 and Table 4-2 shows both the results from both the controlled experiment and the uncontrolled experiment. The controlled environment is a test setup such as using two machines and a router in home network (or private network). In the controlled environment, the messages are never transported over Internet or public network. Thus, we can minimize the chance of unwanted variants affect on the authentication time. Internet is one of the un-controlled environments where we cannot have all controls to maximize and to accurate test results. The test in the un-controlled environment was executed on one machine (a server), is at University of New Hampshire (UNH) and another machine (a client), is in Epsom, NH where is about 27 miles away from UNH over Internet.



**Figure 4-1: Authentication Time for Each Method**

**Table 4-2: 95% Confidence Level Authentication Time for each Method**

Authentication Time [ms] \Method		ANONYMOUS	EAP-PSK	EAP-AES-PSK
Controlled Experiments	Server	88.5±0.5	101±0.5	113±0.7
	Client	27.7±0.4	38.1±0.4	40.4±0.4
Un-Controlled Experiments	Server	290±6	450±27	467±16
	Client	227±3	386±26	395±8

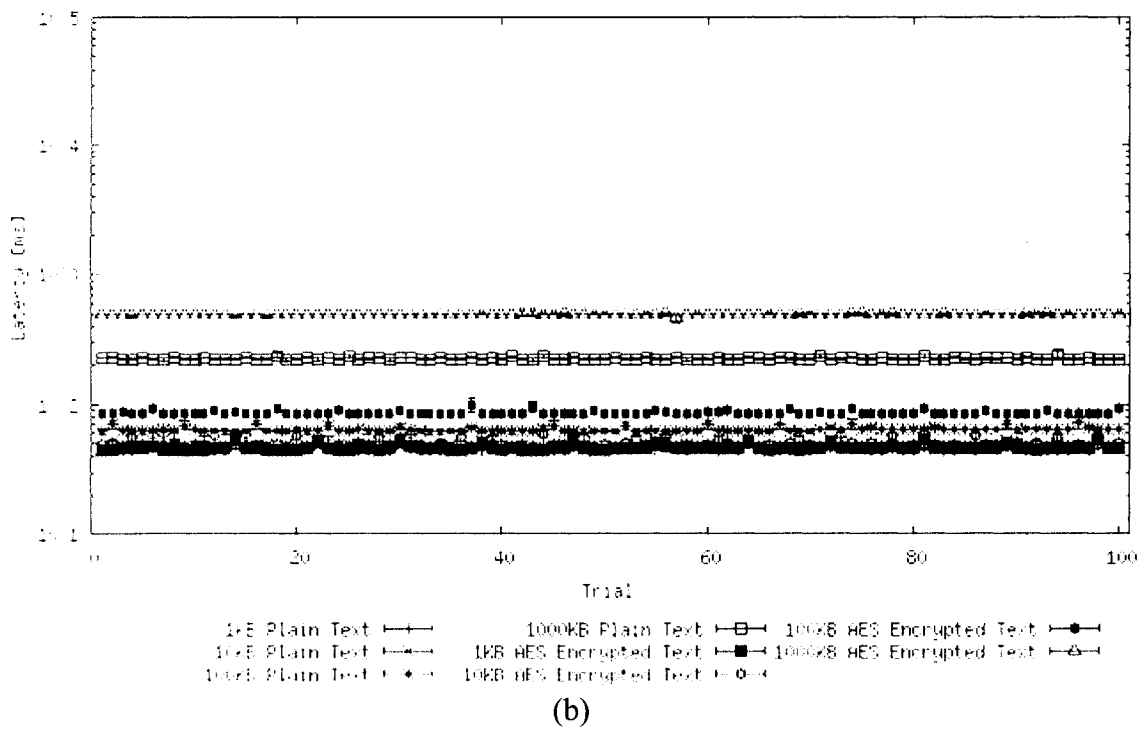
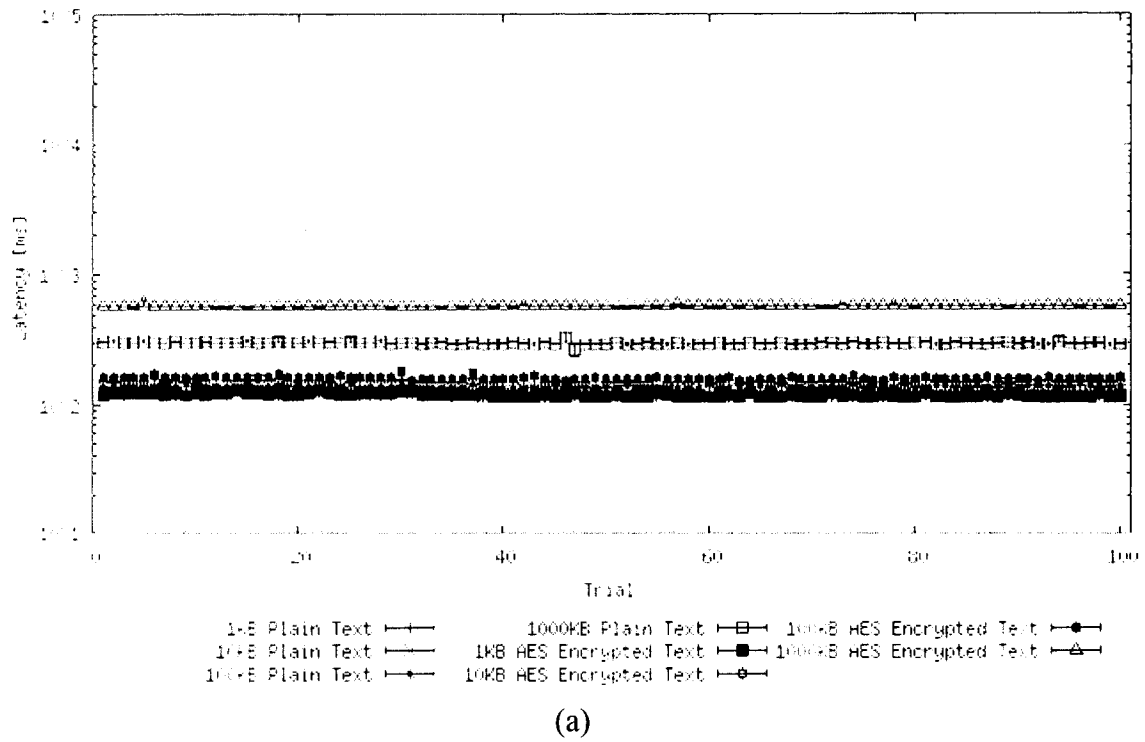
The authentication time in EAP-PSK and EAP-AES-PSK took longer compared to the time in ANONYMOUS because ANONYMOUS only needs 2 message exchanges while EAP-PSK and EAP-AES-PSK requires 4 and 7 message exchanges along with AES encryption.

#### **AES Encryption Time Test:**

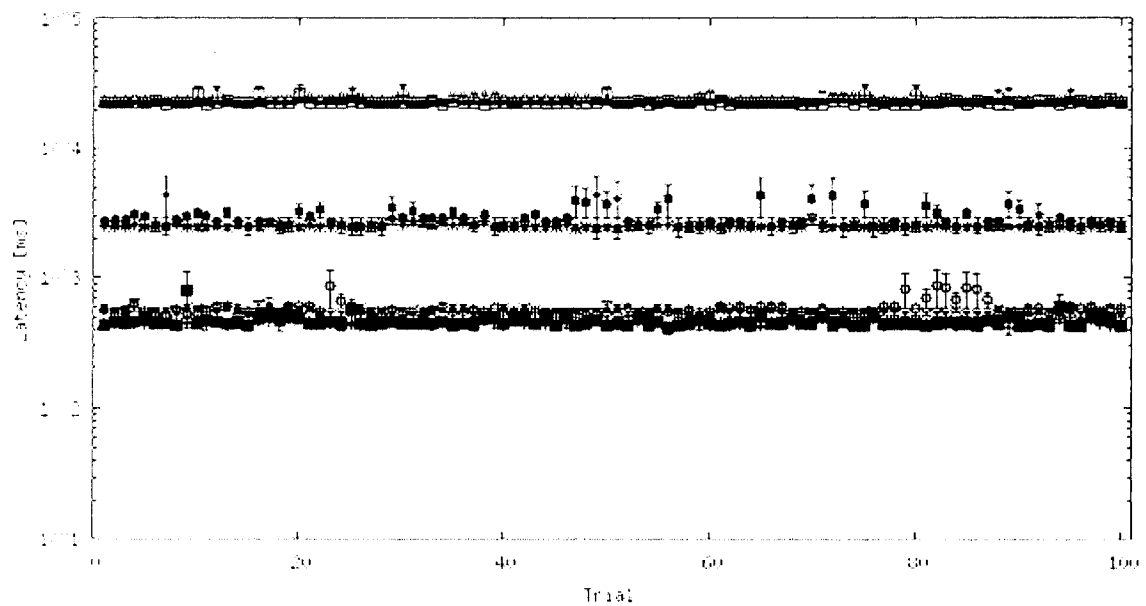
We need to know how much AES encryption time affects on the performance of SASL incorporating EAP-AES-PSK. EAP-AES-PSK, the new proposed EAP method, uses AES encryption to protect data communications after successful authentication.

Figure 4-2 & Table 4-3 and Figure 4-3 & Table 4-4 show the completion time examined on SASL application incorporating EAP-AES-PSK in the controlled environment and the un-controlled environment respectively. The SASL application transfers different sizes of plain text and AES encrypted text files, such as 1 KB, 10 KB, 100 KB and 1000 KB. The text of RFC2800, the one of IETF standards, was randomly selected as a data file for the test and the file was truncated to satisfy the required file sizes; 1 KB, 10 KB, 100 KB and 1000 KB. This completion time tests measured the time from the authentication to the file transfer, which is the sum of the access time and the transfer time.

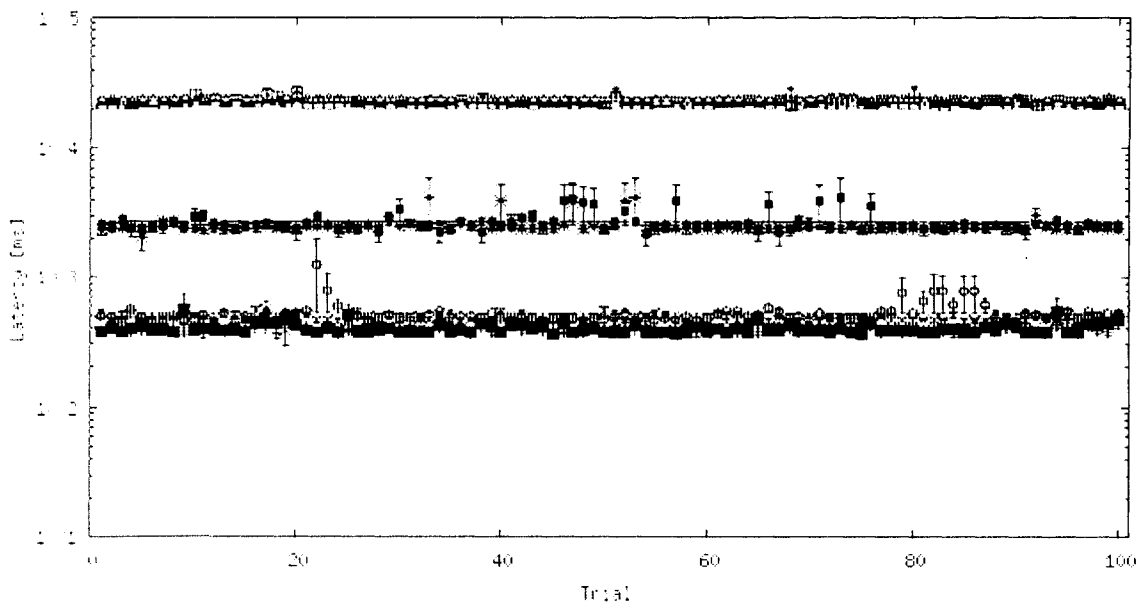
Both (a) and (b) in Figure 4-2 and 4-3 have 800 elements, such as 2 (encrypted/plain text) x 4 (1, 10, 100, and 1000 KB) x 100 (trials). Since the completion time is similar, whether the file is encrypted or not in each different size, we see 4~5 major horizontal lines. As file size increases, Both Figure 4-2 & Table 4-3 (from the controlled experiments) and Figure 4-3 & Table 4-4 (from the un-controlled experiments) show that AES encryption time does not increase much. As following Schneier's AES performance test [21], the test exposes that AES algorithm performs efficiently even in large data, i.e., larger than 1 MB. AES algorithm can be coded in assembly for better performance in where needs to run fast [21]. Thus, SASL file transferring application incorporating EAP-AES-PSK takes an almost comparable amount of time to transfer files from one system to the other as compared to the time to transfer plain text files.



**Figure 4-2:** (a) Server Completion Time in the Controlled Environment,  
(b) Client Completion Time in the Controlled Environment



(a)



(b)

**Figure 4-3:** (a) Server Completion Time in the Un-Controlled Environment,  
(b) Client Completion Time in the Un-Controlled Environment



**Table 4-3: 95% Confidence Level Completion Time in the Controlled Environment**

Completion Time [ms] File size [KB]		1	10	100	1000
Server	Plain Text	119±0.8	121±0.7	138±0.8	301±1.3
	AES Encrypted Text	119±0.7	122±0.8	160±0.9	576±0.9
Client	Plain Text	45.9±0.5	47.9±0.4	63.7±0.5	227±0.8
	AES Encrypted Text	46.2±0.5	49.2±0.5	86.5±0.6	501±1.0

**Table 4-4: 95% Confidence Level Completion Time in the Un-Controlled Environment**

Completion Time [ms] File size [KB]		1	10	100	1000
Server	Plain Text	456±5	540±5	2586±62	22692±172
	AES Encrypted Text	456±9	577±15	2835±74	23363±240
Client	Plain Text	402±5	484±5	2560±72	22556±165
	AES Encrypted Text	402±7	521±15	2646±64	22806±161

## **CHAPTER 5**

# **CONCLUSION**

Authentication and access control of users are key functions to establish secure communication. However, successful authentication does not always occur over insecure network connections. Weak authentication methods are subject to the risks of insecure communication such as dictionary attack, replay attack, and Man-In-The-Middle (MITM) attack.

In this thesis, we demonstrated the incorporation of Extensible Authentication Protocol (EAP) into Simple Authentication and Security Layer (SASL) to enhance the security of SASL and to provide a pathway for easy incorporation of future EAP enhancements into SASL in addition to the establishment of secure communication. Moreover, we proposed a new EAP method, EAP-Advanced Encryption Standard-Pre-Shared Key (EAP-AES-PSK) to provide strong authentication and implemented it on the Cyrus SASL library that is one of the publicly available SASL implementations.

The evaluation of this thesis on the functionality of a SASL application incorporating EAP, the arguments to show how common security risks are addressed, and the performance evaluation shows that SASL incorporating EAP provides stronger security services. The performance test results demonstrated that AES encryption does

not require much more time, and as a result it, this new EAP method, EAP-AES-PSK, can be used in transferring a large sized file.

For the future, EAP-AES-PSK needs a more efficient way to re-authenticate since current re-authentication schemes reduce only 1 round trip. And interoperability with other EAP methods that use other encryptions is an additional concern that warrants further study.

# BIBLIOGRAPHY

- [1] "ADVANCED ENCRYPTION STANDARD (AES)", Federal Information Processing Standards (FIPS) Publication 197, November 26, 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [2] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and Levkowetz, H., "Extensible Authentication Protocol (EAP)," RFC 3748, June 2004.
- [3] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication," <http://www.saunalahti.fi/~asokan/research/mitm.html>, Nokia Research Center, Finland, October 24, 2002.
- [4] Authentication Protocols. Retrieved November 23, 2006, <http://www.comptechdoc.org/independent/networking/protocol/protauthen.html>
- [5] Bersani F., Tschofenig H., "The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) method," RFC 4764, January 2006.
- [6] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions," RFC 3546, June 2003.
- [7] Cheng, P., and Glenn, R., "Test Cases for HMAC-MD5 and HMAC-SHA-1," RFC 2202, Retrieved November 15, 2006, <http://tools.ietf.org/html/rfc2202>
- [8] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0," RFC 2246, November 1998.
- [9] Extensible Authentication Protocol. (2006, November 14). In Wikipedia, The Free Encyclopedia. Retrieved November 15, 2006, [http://en.wikipedia.org/w/index.php?title=Extensible\\_Authentication\\_Protocol&oldid=87776933](http://en.wikipedia.org/w/index.php?title=Extensible_Authentication_Protocol&oldid=87776933)
- [10] Generic Security Services Application Program Interface. (2006, October 31). In Wikipedia, The Free Encyclopedia. Retrieved November 21, 2006, [http://en.wikipedia.org/w/index.php?title=Generic\\_Security\\_Services\\_Application\\_Program\\_Interface&oldid=84773404](http://en.wikipedia.org/w/index.php?title=Generic_Security_Services_Application_Program_Interface&oldid=84773404)
- [11] Hiller, T., Palekar, A., and Zorn, G., "A Container Type for the Extensible Authentication Protocol (EAP)," Retrieved November 06, 2006, <http://bgp.potaroo.net/ietf/all-ids/draft-hiller-eap-tlv-00.txt>

- [12] IEEE Standards for Local and Metropolitan Area Networks: Port-Based Network Access Control, IEEE Std 802.1X-2004, 13 December 2004.
- [13] Kerberos: The Network Authentication Protocol (2006, November 18). Retrieved November 23, 2006, [http://web.mit.edu/kerberos/#what\\_is](http://web.mit.edu/kerberos/#what_is)
- [14] Kim, M., Valcourt, S., and Bartoš, R., "Selecting a Standard Outer Method for EAP," Technical Report 06-01, University of New Hampshire, May 12, 2006.
- [15] Malinen, J., "Host AP driver for Intersil Prism2/2.5/3, hostapd, and WPA Supplicant," Retrieved May 20, 2006, <http://hostap.epitest.fi/>
- [16] Melnikov, A., "The Kerberos V5 ("GSSAPI") SASL mechanism," Retrieved November 15, 2006, <http://www.ietf.org/internet-drafts/draft-ietf-sasl-gssapi-08.txt>
- [17] Melnikov, A. and Zeilenga, K., "Simple Authentication and Security Layer (SASL)," RFC 4422, June 2006.
- [18] Project Cyrus, Carnegie Mellon University, Retrieved November 23, 2006, <http://cyrusimap.web.cmu.edu/>
- [19] Replay attack. (n.d.). Computer Desktop Encyclopedia. Retrieved November 02, 2006, <http://www.answers.com/topic/man-in-the-middle-attack>
- [20] Schneier, B. and Whiting, D., "A Performance Comparison of the Five AES Finalists," Third AES Candidate Conference, 2000.
- [21] Simpson, W. Ed., "The Point-to-Point Protocol (PPP)," RFC 1661 July 1994.
- [22] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)," RFC 1994, August 1996.
- [23] "The Java SASL API Programming and Deployment Guide," Sun Microsystems, 2004, <http://java.sun.com/j2se/1.5.0/docs/guide/security/sasl/sasl-refguide.html>
- [24] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism," RFC 4505, June 2006.

# **APPENDICES**

# APPENDIX A

## ABBREVIATIONS AND ACRONYMS

<b>AAA</b>	Authentication, Authorization, and Accounting	[RFC 3127]
<b>AES</b>	Advanced Encryption Standard	[FIPS 197]
<b>AK</b>	Authentication Key	
<b>CHAP</b>	Challenge Handshake Authentication Protocol	[RFC 1994]
<b>DoS</b>	Denial of Service	
<b>EAP</b>	Extensible Authentication Protocol	[RFC 3748]
<b>EAP-GTC</b>	EAP-Generic Token Card	[RFC 3748]
<b>EAP-PSK</b>	EAP-Pre-Shared Key	[RFC 4764]
<b>EAP-TLS</b>	EAP-Transport Layer Security	[RFC 2716]
<b>EAP-TLS-PSK</b>	EAP- Transport Layer Security-Pre-Shared Key	
<b>EMSK</b>	Extended Master Session Key	
<b>GSSAPI</b>	Generic Security Services Application Program Interface (also GSS-API) [RFC 1508]	
<b>IEEE</b>	Institute of Electrical and Electronics Engineers	
<b>IMAP</b>	Internet Message Access Protocol	
<b>IP</b>	Internet Protocol	
<b>KDK</b>	Key-Derivation Key	
<b>LDAP</b>	Lightweight Directory Access Protocol	
<b>MAC</b>	Message Authentication Code, same as MIC	
<b>MIC</b>	Message Integrity Check, same as MAC (also Message Authentication Code)	
<b>MITM</b>	Man-In-The-Middle attack	
<b>MSK</b>	Master Session Key	
<b>PAP</b>	Password Authentication Protocol	
<b>POP</b>	Post Office Protocol	
<b>PPP</b>	Point-to-Point Protocol	
<b>PRF</b>	Pseudo Random Function	
<b>PSK</b>	Pre-Shared Key	
<b>SASL</b>	Simple Authentication and Secure Layer	[RFC 4422]
<b>SMTP</b>	Simple Mail Transfer Protocol	
<b>SQL</b>	Structured Query Language	
<b>TEK</b>	Transient EAP Key	
<b>TLS</b>	Transport Layer Security	[RFC 4346]
<b>TLV</b>	Type-Length-Value	
<b>VPN</b>	Virtual Private Network	
<b>WPA</b>	Wi-Fi Protected Access	

# APPENDIX B

## EXPERIMENTS/USER GUIDES

This document is written to help users to install and to use CYRUS (SASL library) in the simplest way as well as to incorporate Extensible Authentication Protocol (EAP) into SASL. CYRUS could be integrated with DB such as SASLdb (gdbm, Sleepycat, ndbm), SQL (mySQL and PostgreSQL v7.2 or higher) to enable userID/password.

### B.1 Cyrus Installation

- \* Note that some of the following steps require users to be a root.
- \*\* If you want to have SASL file transfer application incorporating EAP, do all 10 steps. Otherwise, do only first seven steps.

- Step 1 download cyrus-sasl-2.1.22
- Step 2 untar cyrus-sasl-2.1.22
- Step 3 cd (directory it was untarred into) i.e., cyrus-sasl-2.1.22
- Step 4 ./configure
- Step 5 make
- Step 6 make install
- Step 7 ln -s /usr/local/lib/sasl2 /usr/lib/sasl2
- Step 8 untar cyrus-sasl-2.1.22-eap.tar
- Step 9 replace all files include/, lib/, plugins/, and sample/ directory with files in cyrus-sasl-2.1.22-eap/ directory
- Step 10 re-compile



## B.2 Cyrus usage of ANONYMOUS mechanism with file transfer

### **\*\* Server side**

Step 1 open terminal  
Step 2 `cd cyrus-sasl-2.1.22/sample`  
Step 3 `./server`

Now a server waits for client connection.

### **\*\* Client side**

Step 1 open terminal  
Step 2 `cd cyrus-sasl-2.1.22/sample`  
Step 3 `./client -m ANONYMOUS host_address`  
Step 4 enter user account i.e., "mkim" (from mkim@localhost)  
Step 5 enter file name

After client transfers a file, it will terminate the program.

In order to use other authentication methods such as CRAM-MD5 and DIGEST-MD5,  
type `./client -m method_name host_address`.

## B.3 Cyrus Password Setup (using sasldb)

### **Password setup using SASLDB v.2**

Step 1 `/usr/sbin/saslpasswd2 userID`  
Step 2 enter user password  
User ID and password will be saved on `/etc/sasldb2`  
The script, `/usr/sbin/sasldblistusers2`, displays all users in the SASLDB.

### **Password setup using SASLDB v.1**

Step 1 `/usr/sbin/saslpasswd userID`  
Step 2 enter user password  
User ID and password will be saved on `/etc/sasldb`  
The script, `/usr/sbin/sasldblistusers`, displays all users in the SASLDB.

## B.4 Add Mechanism Plugins

The following example explains how to add EAP-PSK. If you want to add other methods, replace EAP-PSK with the method name.

### Step 1 Implement the following functions in `plugins/` directory:

#### Step 1.1 Both client/server sides: add `plugin_id`

```
i.e., static const char plugin_id[] = "$Id: eappsk.c, v
1.11 yyyy/mm/dd hh:mm:ss mel Exp $"
```

#### Step 1.2 Server sides

##### Step 1.2.1 initialize server mechanism - `mechName_server_mech_new`

```
i.e., static int eappsk_server_mech_new(
    void *glob_context __attribute__((unused)),
    sasl_server_params_t *sparams,
    const char *challenge __attribute__((unused)),
    unsigned challen __attribute__((unused)),
    void **conn_context);
```

##### Step 1.2.2 process server mechanism - `mechName_server_mech_step`

```
i.e., static int eappsk_server_mech_step(
    void *conn_context __attribute__((unused)),
    sasl_server_params_t *params,
    const char *clientin,
    unsigned clientinlen,
    const char **serverout,
    unsigned *serveroutlen,
    sasl_out_params_t *oparams);
```

##### Step 1.2.3 fill up server mechanism plugins information - `mechName_server_plugins`

```
i.e., static sasl_server_plug_t eappsk_server_plugins[] =
{
    {
        "EAPPSK", /* mech_name */
        0, /* max_ssf */
        SASL_SEC_NOANONYMOUS, /* security_flags */
        SASL_FEAT_WANT_CLIENT_FIRST
        | SASL_FEAT_ALLOWS_PROXY, /* features */
        NULL, /* glob_context */
        &eappsk_server_mech_new, /* mech_new */
        &eappsk_server_mech_step, /* mech_step */
    }
}
```

```

        NULL,                /* mech_dispose */
        NULL,                /* mech_free */
        NULL,                /* setpass */
        NULL,                /* user_query */
        NULL,                /* idle */
        NULL,                /* mech_avail */
        NULL                 /* spare */
    }
};

```

**Step 1.2.4 initialize server mechanism plugin - mechName\_server\_plug\_init**

```

i.e., int eappsk_server_plug_init(
        const sasl_utils_t *utils,
        int maxversion,
        int *out_version,
        sasl_server_plug_t **pluglist,
        int *plugcount);

```

**Step 1.2.5 de-initialize server mechanism - mechName\_server\_mech\_dispose**

**Step 1.2.6 other functions: mechName\_server\_mech\_free, mechName\_server\_mech\_avail**

### Step 1.3 Client side

**Step 1.3.1 initialize client mechanism - mechName\_client\_mech\_new**

```

i.e., static int eappsk_client_mech_new(
        void *glob_context __attribute__((unused)),
        sasl_client_params_t *params,
        void **conn_context);

```

**Step 1.3.2 process client mechanism - mechName\_server\_mech\_step**

```

i.e., static int eappsk_client_mech_step(
        void *conn_context,
        sasl_client_params_t *params,
        const char *serverin __attribute__((unused)),
        unsigned serverinlen __attribute__((unused)),
        sasl_interact_t **prompt_need,
        const char **clientout,
        unsigned *clientoutlen,
        sasl_out_params_t *oparams);

```

**Step 1.3.3 fill up client mechanism plugins information - mechName\_clientn\_plugins**

```

i.e., static sasl_client_plug_t eappsk_client_plugins[] =
{
    {
        "EAPPSK",                /* mech_name */
        0,                       /* max_ssf */
        SASL_SEC_NOANONYMOUS,    /* security_flags */
        SASL_FEAT_WANT_CLIENT_FIRST
        | SASL_FEAT_ALLOWS_PROXY, /* features */
        NULL,                    /* required_prompts */
        NULL,                    /* glob_context */
        &eappsk_client_mech_new,  /* mech_new */
        &eappsk_client_mech_step, /* mech_step */
        &eappsk_client_mech_dispose, /* mech_dispose */
        NULL,                    /* mech_free */
        NULL,                    /* idle */
        NULL,                    /* spare */
        NULL                     /* spare */
    }
};

```

**Step 1.3.4 initialize client mechanism plugin - mechName\_client\_plug\_init**

```

i.e., int eappsk_client_plug_init(sasl_utils_t *utils,
                                int maxversion,
                                int *out_version,
                                sasl_client_plug_t **pluglist,
                                int *plugcount);

```

**Step 1.3.5 de-initialize client mechanism - mechName\_client\_mech\_dispose**

**Step 1.3.6 other function: mechName\_client\_mech\_free**

## **Step 2 Call the following functions in application codes**

**Step 2.1 Server side**

**Step 2.1.1 go to the line sasl\_server\_init() is called**

**Step 2.1.2 call the fuction sasl\_server\_add\_plugin()**

```

i.e., sasl_server_add_plugin("EAPPSK",
                             &eappsk_server_plug_init);

```

**Step 2.1.3 include the new mechanism**

```

i.e., #include "../plugins/eappsk.c"

```

Step 2.2 Client side: Software engineers must call the following functions in application codes where locates in the directory sample/

Step 2.2.1 go to the line `sasl_client_init()` is called

Step 2.2.2 call the fuction `sasl_client_add_plugin()`

i.e., `sasl_client_add_plugin("EAPPSK",  
                                    &eappsk_client_plug_init);`

Step 2.2.3 include the new mechanism

i.e., `#include "../plugins/eappsk.c"`

### Step 3 Modify the file configure in Cyrus directory

Step 3.1 search `--enable-plain`

Step 3.2 add the following code segment before the line you found

```
##### EAPPSK #####

# Check whether --enable-eappsk or --disable-eappsk
was given.
if test "${enable_eappsk+set}" = set; then
    enableval="$enable_eappsk"
    eappsk=$enableval
else
    eappsk=yes
fi;

EAPPAK_LIBS=""
if test "$eappsk" != no; then
    if test "$cmu_have_crypt" = yes; then
        EAPPSK_LIBS=$LIB_CRYPT
    fi
fi

echo "$as_me:$LINENO: checking EAPPSK" >&5
echo $ECHO_N "checking EAPPSK... $ECHO_C" >&6
if test "$eappsk" != no; then
    echo "$as_me:$LINENO: result: enabled" >&5
echo "${ECHO_T}enabled" >&6
    SASL_MECHS="$SASL_MECHS libeappsk.la"
    if test "$enable_static" = yes; then
        SASL_STATIC_OBJS="$SASL_STATIC_OBJS eap-psk.o"
```

```
SASL_STATIC_SRCS="$SASL_STATIC_SRCS ../plugins/eap-
psk.c"
```

```
cat >>confdefs.h <<\_ACEOF
#define STATIC_EAPPSK
_ACEOF

fi
else
echo "$as_me:$LINENO: result: disabled" >&5
echo "${ECHO_T}disabled" >&6
fi
#####
```

**Step 4** Add new mechanisms on the file, **Makefile.am**, in the directory **plugins/**

**Step 5** Make sure that you re-compile for all the changes

**Step 6** Make sure that the directory **/usr/lib/sasl2/** contains the new mechanism library

```
i.e., locate libeappsk.la
locate libeappsk.so
```