

**DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES**

**SECURE DATA TRANSMISSION OVER
WIRELESS NETWORKS**

by
Bakytbek ESHMURZAEV

**January, 2012
İZMİR**

SECURE DATA TRANSMISSION OVER WIRELESS NETWORKS

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering, Computer Engineering Program**

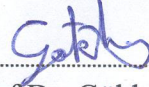
**by
Bakytbek ESHMURZAEV**

January, 2012

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled "SECURE DATA TRANSMISSION OVER WIRELESS NETWORKS" completed by BAKYTBEK ESHMURZAEV under supervision of ASST.PROF.DR. GÖKHAN DALKILIÇ and we certify that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.



Asst.Prof.Dr. Gökhan DALKILIÇ

Supervisor



Asst.Prof.Dr. Tugkan TUĞLULAR

(Jury Member)



Prof. Dr. Yalçın ÇEBİ

(Jury Member)



Prof.Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to show my gratitude to my supervisor Asst. Prof. Dr. Gökhan DALKILIÇ for his guidance, suggestions and patience.

Furthermore, I would like to thank to one of the authors of EAP-FAST protocol Hao Zhou for his guidance on learning the protocol details and I am also thankful to Tigran S. Avanesov and Laurent Vigneron who helped me in modeling the protocol in HLPSL.

I would also thankful to all Ph.D. students for their help.

Finally, I would like to thank to my friends and to my family for their great support and love.

Bakytbek ESHMURZAEV

SECURE DATA TRANSMISSION OVER WIRELESS NETWORKS

ABSTRACT

Authentication and encryption are the core security concepts of Wireless LAN. Today, very strong security mechanisms for WLAN do exist. If proper WLAN security solutions are deployed, a wireless network can be as secure as the wired network. An 802.1X/EAP framework allows a variety of specific methods to be used for the authentication and key management procedures. There are two major sets of EAP-methods, which are password-based and certificate-based. The password-based EAP-types provide lightweight processing and are very convenient. But many of them are susceptible to the offline dictionary attacks, and hence considered weak. On the other hand, the certificate-based methods provide strong security as well as allow password-based authentication methods to be used. The certificate-based methods achieve these security properties using Transport Layer Security (TLS) Handshake protocol that establishes authenticated and encrypted tunnel. Within tunnel, password-based methods can run securely. The significant downside of certificate-based methods is the requirement of Public Key Infrastructure (PKI) which is costly to implement and hard to manage. This research's aim is an analysis of TLS-based EAP protocols used in WLAN. We have chosen only wide deployed RFC-based EAP types because of their availabilities and standards based property. We mainly focus on the EAP-FAST protocol because of its attracting security features. The EAP-FAST protocol differs from other TLS-based EAP types on using shared secret keys instead of certificates, thus significantly increasing performance. EAP-FAST provides not only the same security level as other strong TLS-based methods, but also convenience and efficiency by using PACs. We validated different authentication scenarios of EAP-FAST protocol using an AVISPA model checker.

Keywords: Wireless LAN Security, 802.1X, Extensible Authentication Protocol (EAP), 802.11 Authentication and Key Management, Tunnel-based EAP methods, AVISPA.

KABLOSUZ AĞLAR ÜZERİNDEN GÜVENLİ VERİ İLETİMİ

ÖZ

Kimlik doğrulama ve şifreleme Kablosuz Yerel Alan Ağları'nın (KYAA) temel güvenlik kavramlarıdır. Günümüzde KYAA için çok güçlü güvenlik çözümleri mevcuttur. Eğer uygun KYAA güvenlik çözümleri uygulanırsa, kablosuz ağ da kablolu ağ kadar güvenli olabilir. 802.1X/EAP taslağı kimlik doğrulama ve anahtar yönetimi prosedürlerinde kullanılmak üzere çeşitli özellikli metotlara izin verir. EAP metotları şifre-bazlı ve sertifika-bazlı olarak iki ana gruba ayrılırlar. Şifre-bazlı EAP metotları hafif işlerde kullanılır ve çok elverişlidirler. Ama birçoğu şifre-bazlı metotlar çevrimdışı sözlük saldırılarına elverişlidir ve dolayısıyla zayıftırlar. Diğer taraftan, sertifika-bazlı metotlar güçlü güvenlik sağlarlar, aynı zamanda zayıf sayılan şifre-bazlı metotların da kullanılmasına izin verirler. Sertifika-bazlı metotlar bu güvenliği doğrulanmış ve şifrelenmiş tünel oluşturan TLS protokolünü kullanarak elde ederler. Bu tünelde şifre-bazlı metotlar güvenli olarak çalışırlar. Sertifika-bazlı metotların kötü yanı uygulaması pahalı ve yönetimi zor olan Açık Anahtar Altyapısına ihtiyaç duymalarıdır. Bu çalışmanın amacı KYAA'nda kullanılan TLS-bazlı EAP protokollerinin analizidir. Sadece geniş çaplı kullanılan RFC tabanlı EAP tiplerini seçmemizin nedeni kullanılabilirlikleri ve standart tabanlı özelliğidir. Cezbedici güvenlik özelliklerinden dolayı özellikle EAP-FAST protokolünü odaklandık. EAP-FAST protokolü diğer TLS-bazlı EAP tiplerinden sertifikalar yerine paylaşılmış gizli anahtar kullanımlarından dolayı ayrılmaktadır ki bu da önemli şekilde performansı arttırmaktadır. EAP-FAST sadece diğer güçlü TLS-bazlı metotlar gibi aynı güvenlik seviyesini sağlamamakta aynı zamanda PAC'leri kullanarak elverişliliği ve verimliliği de sağlamaktadır. Bu çalışmada, EAP-FAST metodunun değişik doğrulama mekanizmalarını AVISPA model denetçisi ile onayladık.

Anahtar sözcükler: Kablosuz yerel ağ güvenliği, 802.1X, Genişletilebilir Kimlik Doğrulama Protokolü (EAP), 802.11 Kimlik Doğrulama ve Anahtar Yönetimi, Tünel-bazlı EAP metotları, AVISPA.

CONTENTS

| | Page |
|--|-------------|
| M.SC THESIS EXAMINATION RESULT FORM..... | II |
| ACKNOWLEDGMENTS | III |
| ABSTRACT..... | IV |
| ÖZ | V |
| | |
| CHAPTER ONE - INTRODUCTION | 1 |
| | |
| 1.1 Evolution of WLAN Security..... | 3 |
| 1.2 RSNA Establishment..... | 6 |
| 1.3 Organization of Thesis | 7 |
| | |
| CHAPTER TWO - AUTHENTICATION AND KEY MANAGEMENT | 8 |
| | |
| 2.1 Stage 1: Discovery of Security Capabilities..... | 10 |
| 2.2 Stages 2 and 3: 802.11 Authentication and Association | 12 |
| 2.2.1 802.11 Authentication Methods..... | 13 |
| 2.2.1.1 Open System Authentication vs. Shared Key Authentication | 13 |
| 2.2.2 802.11 Association | 15 |
| 2.3 Stage 4: An 802.1X/EAP Authentication..... | 15 |
| 2.3.1 The 802.1X Standard | 15 |
| 2.3.2 Extensible Authentication Protocol (EAP)..... | 18 |
| 2.3.3 EAP Carrier Protocols | 19 |
| 2.3.3.1 EAPOL Protocol | 19 |
| 2.3.3.2 RADIUS Protocol | 20 |
| 2.3.4 EAP Methods..... | 21 |
| 2.4 Stages 5, 6 and 7: Key Management | 23 |
| 2.4.1 RSNA Key Hierarchy | 23 |
| 2.4.2 The Four-Way Handshake..... | 26 |

| | |
|--|-----------|
| 2.4.3 The Group Key Handshake | 27 |
| 2.5 Stage 8: Secure Data Communication..... | 28 |
| CHAPTER THREE - TLS-BASED EAP METHODS | 36 |
| 3.1 TLS-Based EAP Methods Overview | 36 |
| 3.2 EAP-TLS | 38 |
| 3.3 EAP-TTLS..... | 39 |
| 3.4 PEAP | 40 |
| 3.5 EAP-FAST | 41 |
| 3.5.1 PAC Types..... | 42 |
| 3.5.2 Dynamic PAC Provisioning | 43 |
| 3.5.3 EAP-FAST Provisioning Modes | 45 |
| 3.5.4 MITM on Tunnel-Based EAP Methods | 46 |
| 3.5.5 EAP-FAST MiTM Attack Protection..... | 47 |
| 3.5.6 Summary of EAP-FAST Features | 48 |
| 3.6 Comparison of TLS-Based EAP Methods | 48 |
| CHAPTER FOUR - THE AVISPA TOOL..... | 50 |
| 4.1 The High Level Protocol Specification Language (HLPSL) | 51 |
| 4.2 The Dolev-Yao Intruder | 55 |
| 4.3 The Back-End Analyzers..... | 56 |
| 4.4 The SPAN Tool..... | 57 |
| 4.5 Summary of AVISPA Features | 59 |
| 4.6 AVISPA Modeling Limitations..... | 60 |
| 4.7 AVISPA Usage Recommendations..... | 61 |
| CHAPTER FIVE - VALIDATION OF PROTOCOLS..... | 62 |
| 5.1 Dynamic Provisioning using EAP-FAST..... | 63 |
| 5.1.1 Server-Authenticated Provisioning Mode | 63 |

| | |
|--|------------|
| 5.1.2 Server-Unauthenticated Provisioning Mode..... | 67 |
| 5.2 EAP-FAST Authentication Mechanisms | 70 |
| 5.2.1 Tunnel Establishment with Tunnel PAC | 71 |
| 5.2.2 Inner Authentication with User-Authorization PAC | 72 |
| 5.3 The Four-Way Handshake Protocol | 73 |
| | |
| CHAPTER SIX - CONCLUSION | 75 |
| | |
| REFERENCES | 77 |
| | |
| APPENDIX A - AVISPA FAQ..... | 84 |
| | |
| APPENDIX B - EAP-FAST AUTHENTICATION | 86 |
| B.1 Tunnel PAC Usage | 86 |
| B.2 User Authorization PAC Usage..... | 91 |
| | |
| APPENDIX C - DYNAMIC PROVISIONING USING EAP-FAST..... | 95 |
| C.1 Server-Authenticated Provisioning..... | 95 |
| C.2 Server-Unauthenticated Provisioning | 100 |
| | |
| APPENDIX D - THE FOUR-WAY HANDSHAKE PROTOCOL | 108 |

CHAPTER ONE

INTRODUCTION

Nowadays, wireless is being offered everywhere and it is becoming more and more popular. The main reasons of preferring wireless network to wired network are (Dengg & others, 2009):

- Flexibility
- Mobility
- High productivity
- Easy of deployment
- Expandability
- Lower cost

Contrary to the aforementioned features, wireless networking introduces new security issues that require new security solutions. Here is some example:

Wireless technology works only at the Physical layer and the Media Access Control (MAC) sublayer of the Data-Link layer of the OSI model. The Logical Link Control (LLC) sublayer of the Data-Link layer and other upper layers are identical for all 802 - based networks. The physical layer for wireless is air (Figure 1.1). So, attackers do not need physical access to get data that is being transmitted freely and openly in the air. Thus strong encryption is needed to ensure data privacy (Coleman, Westcott, Harkins & Jackman, 2010).

Most wireless networks provide a portal into wired networks. Portals have to provide strong authentication solution, which allows to pass only authorized users to the network resources.

Today, very strong security mechanisms for wireless local area networks (WLAN) do exist. If proper WLAN security solutions are deployed, a wireless network can even be more secure than the wired network. For instance, in small

office/home office (SOHO) environments, WPA2-Personal mechanism should be used with strong passwords in order to secure the network, where enterprise corporate wireless networks should be secured with WPA2-Enterprise mechanism. Another example, there is no security at most Wi-Fi hot spots such that airports, cafe's, metro, hotels. Such networks can be secured with VPN technology and Captive portals. The Virtual Private Networks (VPN) provides data privacy for remote access while Captive portals provide authentication. We can give many similar examples, and it is certain that with proper implementation of the security architectures, without doubt, we can be sure about the security of wireless network.

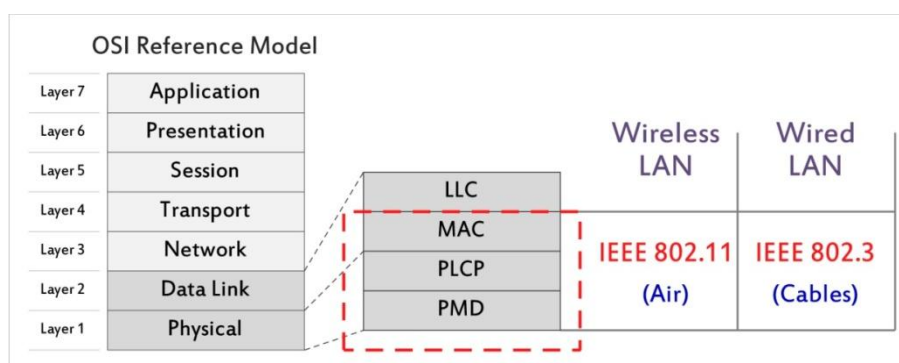


Figure 1.1 Wireless operating layers (Institute of Electrical and Electronics Engineers [IEEE], 2007).

Coleman, Westcott, Harkins & Jackman (2010) lists the five major wireless security components (Figure 1.2):

- Data Confidentiality
- Authentication
- Segmentation
- Monitoring
- Policy

Among above components, authentication and data confidentiality, which we analyzed in this research, are most important.

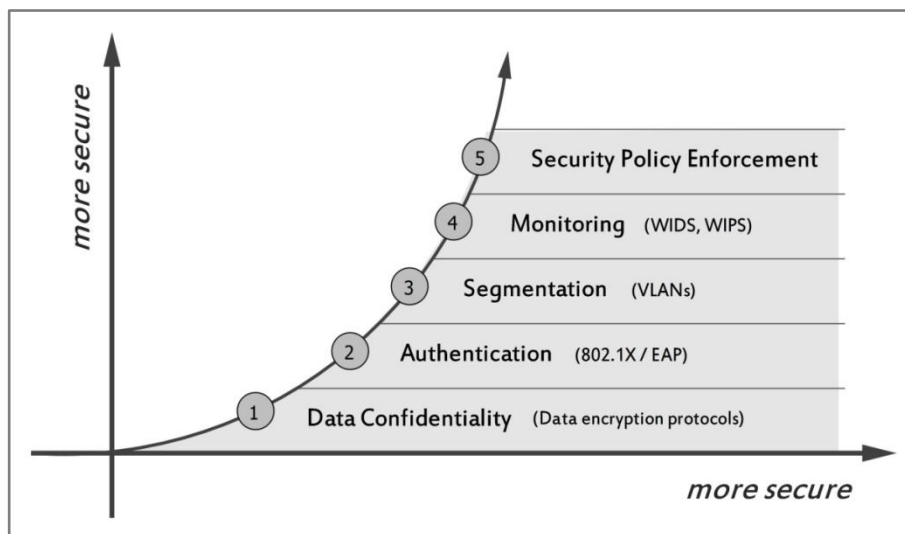


Figure 1.2 Major wireless security components.

1.1 Evolution of WLAN Security

The original 802.11 standard was published in June 1997 as IEEE Std. 802.11-1997, and it defined an encryption protocol called Wired Equivalent Privacy (WEP) and two methods of authentication: Open System authentication and Shared Key authentication. These methods provided the authentication, confidentiality and integrity of WLAN in the past (Coleman & Westcott, 2009). Although now there exists much better and faster methods, the legacy methods are also being used because of legacy hardware which are not capable to support new authentication and confidentiality methods. These legacy security methods have been deprecated except Open System authentication. The deprecated methods should be avoided to use because of their weaknesses.

In 2003, the Wi-Fi Alliance introduced the Wi-Fi Protected Access (WPA) certification as a snapshot of the not-yet-released 802.11i amendment. WPA introduced new 802.1X/EAP authentication and TKIP/RC4 dynamic encryption-key generation methods (Coleman, Westcott, Harkins & Jackman, 2010). Temporal Key Integrity Protocol (TKIP), uses the RC-4 stream cipher algorithm. It was basically an enhancement of WEP encryption and was considered just an interim solution. In 2008, some flaws were found in TKIP/RC4.

In 2004, the 802.11i amendment was ratified by the IEEE and published as IEEE Std. 802.11i-2004. The same year, the Wi-Fi Alliance introduced a more complete implementation of the 802.11i amendment which is referred to as Wi-Fi Protected Access 2 (WPA2) certification. The 802.11i amendment was one of the most important enhancements to the original 802.11 standard. The amendment fully defined a robust security network (RSN) with stronger encryption and better authentication methods. The major enhancement of the amendment was a stronger encryption method called Counter Mode with Cipher Block Chaining Message Authentication Code Protocol (CCMP), which uses the Advanced Encryption Standard (AES) algorithm. The encryption method is often abbreviated as CCMP/AES or just CCMP. The 802.11i amendment also defines an optional encryption method TKIP/RC4 (Figure 1.3) (IEEE, 2004b).

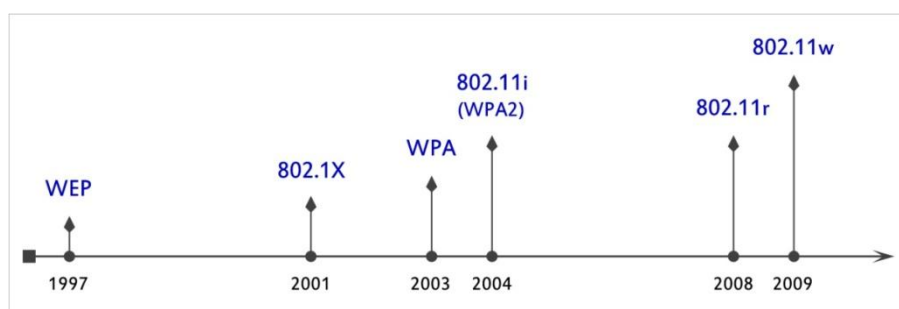


Figure 1.3 The 802.11 security timeline.

Both WPA and WPA2 have two versions:

- *WPA/WPA2-Personal* defines security mechanisms for a Small Office/Home Office (SOHO) environment.
- *WPA/WPA2-Enterprise* defines security mechanisms for enterprise corporate networks.

The main differences between these versions are authentication methods. IEEE 802.1X authorization framework or (PSKs). An IEEE 802.1X/EAP authentication method used within WPA/WPA2-Enterprise while preshared key (PSK) based authentication is used within WPA/WPA2-Personal.

In June 2007, IEEE published new IEEE Std. 802.11-2007 standard which includes eight amendments. The 802.11i security amendment is also now part of the 802.11-2007 standard. All aspects of the 802.11i ratified security amendment can be found in clause 8 of the 802.11-2007 standard (Figure 1.4). The 802.11-2007 standard as the most current guideline to provide operational parameters for WLANs (Table 1.1) (IEEE, 2007).

Table 1.1 The 802.11 standards and certifications (Coleman, Westcott, Harkins & Jackman, 2010).

| 802.11 Standard | Wi-Fi Alliance Certification | Authentication Method | Encryption Method | Cipher | Key Generation |
|-----------------|------------------------------|----------------------------|-------------------------------------|------------|----------------|
| 802.11-1997 | | Open system, Shared Key | WEP | RC4 | Static |
| | WPA-Personal | WPA PSK | TKIP | RC4 | Dynamic |
| | WPA-Enterprise | 802.1X/EAP | TKIP | RC4 | Dynamic |
| 802.11-2007 | WPA2-Personal | WPA2 PSK | CCMP (mandatory) TKIP (optional) | AES RC4 | Dynamic |
| 802.11-2007 | WPA2-Enterprise | 802.1X/EAP | CCMP (mandatory) TKIP (optional) | AES RC4 | Dynamic |

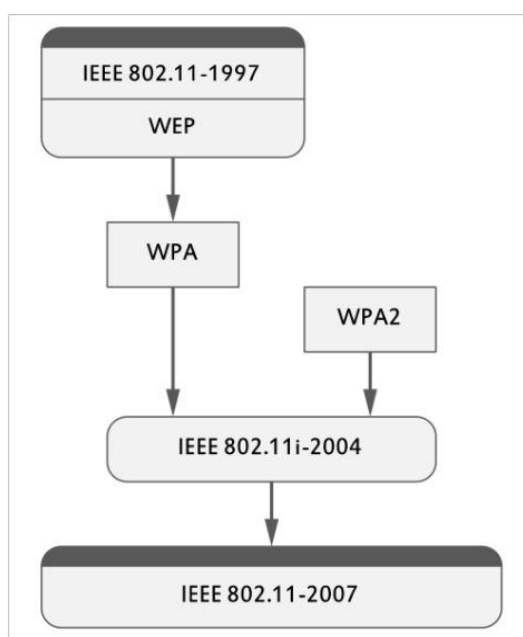


Figure 1.4 The 802.11 security evolution.

The 802.11-2007 standard defines a robust security network (RSN) and robust security network associations (RSNAs). RSNA is an association, in which two stations authenticate and associate with each other as well as create dynamic encryption keys that are unique between those two stations. A robust security network (RSN) is a network that allows for the creation of only robust security network associations (RSNAs). In RSN, CCMP/AES encryption is the mandated encryption method, while TKIP/RC4 is an optional encryption method. It is also possible to create pre-robust security network associations (pre-RSNAs) using legacy security methods, defined in the 802.11-1997 standard, in the same basic service set (BSS) along with RSN-security defined mechanisms. Such networks referred to as Transition Security Networks (TSN). The summary of security mechanisms used in WLANs is shown in Figure 1.5.

1.2 RSNA Establishment

RSNA establishment procedure consists of 802.1X authentication and key management protocol known as the Four-Way Handshake. RSNA establishment procedure involves three entities: the wireless station which may be laptop or PDA, access point and authentication server that is typically RADIUS server. A successful RSNA established means that the station and the access point verified each other's identity and derived some keys for secure data communication with each other. The RSNA establishment stages in enterprise network may be listed as follows:

- Discovery of the network and its capabilities
- Open System authentication and association to the network
- 802.1X/EAP Authentication
- Generation of Master and Temporal keys
- Secure data communication

In SOHO environments, where there is no RADIUS server, preshared keys will be used in generation of Master keys. Thus the 802.1X/EAP authentication step will be omitted.

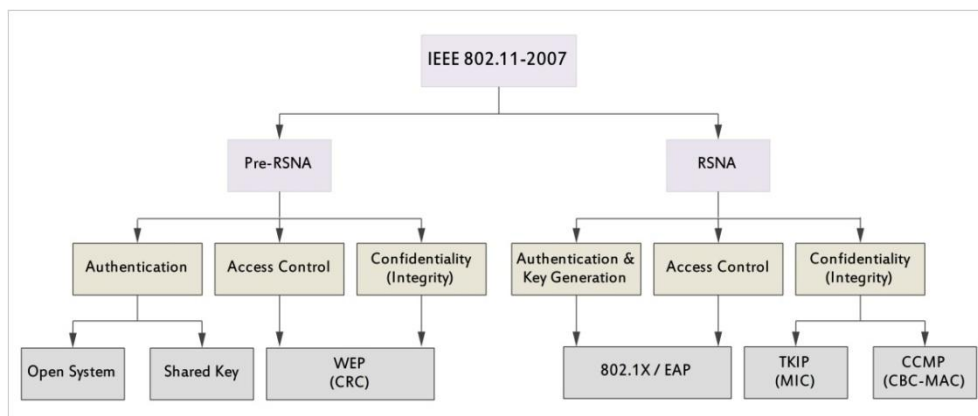


Figure 1.5 The 802.11-2007 standards security.

This research discusses RSNA establishment procedures in infrastructure networks and analyzes tunnel-based Extensible Authentication Protocols (EAP) which are used within 802.1X/EAP framework (Figure 1.5). We have chosen only wide deployed RFC-based EAP types because of their availabilities and standards based property. We mainly focus on the EAP-FAST protocol because of its attracting security features. The EAP-FAST protocol differs from other tunnel-based EAP types on using shared secret keys instead of certificates, thus significantly increasing performance. EAP-FAST provides not only the same security level as other strong tunnel-based methods, but also convenience and efficiency by using Protected Access Credentials (PACs). We validated different authentication scenarios of the EAP-FAST protocol and the four-way handshake key management protocol using an Automated Validation of Internet Security Protocols and Applications (AVISPA) model-checker.

1.3 Organization of Thesis

This research consists of six chapters. Chapter two describes the RSNA Establishment procedures. Chapter three discusses and compares the TLS-based EAP methods. Chapter four introduces the AVISPA tool. Chapter five focuses on validation of protocols and analyzes the output results. Finally, chapter six concludes the research.

CHAPTER TWO

AUTHENTICATION AND KEY MANAGEMENT

This chapter fully focuses on RSNA establishment procedures. RSNAs established using authentication and key management (AKM) services which is defined in the 802.11-2007 standard. The AKM services consist of a set of algorithms which require both authentication processes and the generation and management of encryption keys. Many of these algorithms are non-IEEE-802 protocols that were defined by other standards organizations, such as the Internet Engineering Task Force (IETF). An authentication and key management protocol (AKMP) can be either a preshared key (PSK) or an EAP protocol used during 802.1X authentication. The main goals of 802.1X/EAP are the validation of stations' credentials (authentication) and granting access for the station to network resources (authorization). Although authentication and encryption have different goals and are different processes, they are linked together in AKM services. Authorization is not finalized until encryption keys are created and encryption keys cannot be created without authentication.

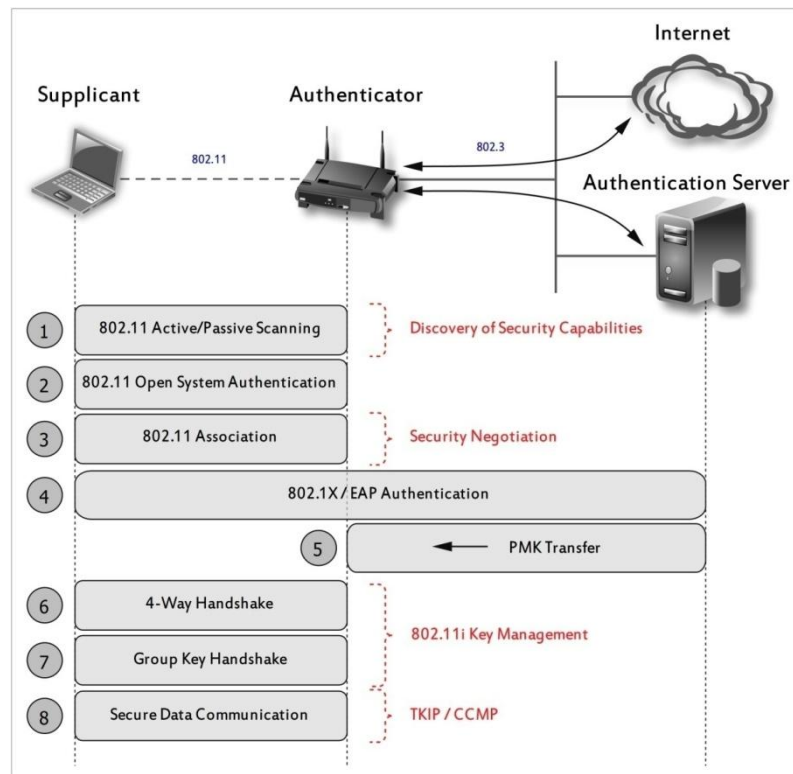


Figure 2.1 The AKM operations within enterprise.

In enterprise network, where the 802.1X/EAP authentication solution is used, AKM operations will be as shown in Figure 2.1.

In SOHO environments, generally there is no use of the 802.1X/EAP authorization framework, the AKM procedures will look like as shown in Figure 2.2. In this environment, preshared key becomes the Master key which is consequently used in derivation of data encryption/decryption keys.

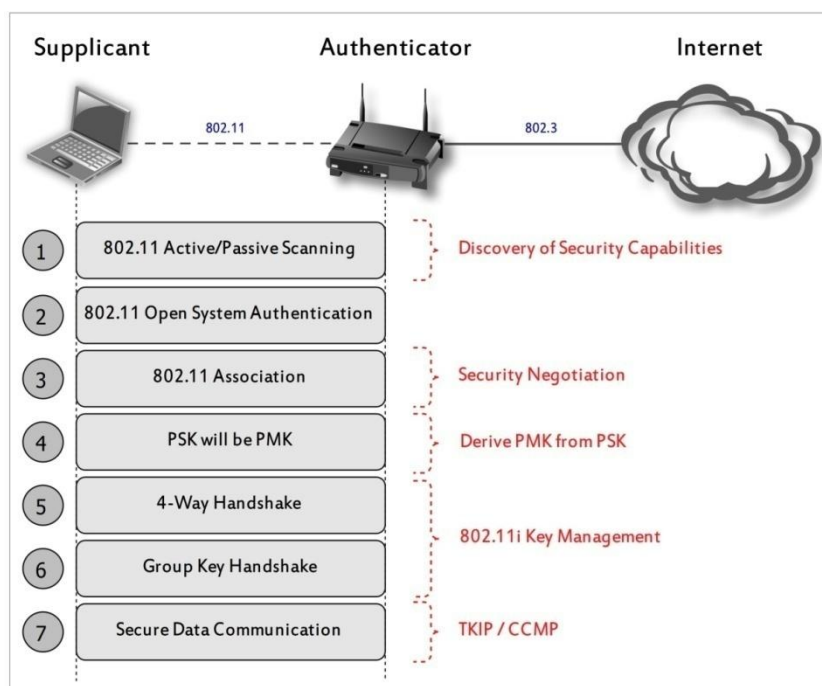


Figure 2.2 The AKM operations within SOHO.

Before discussing details in each stage of AKM process, it is important to understand some concepts of WLAN such as BSS, IBSS and ESS.

WLAN operates in *ad-hoc mode* or *infrastructure mode*.

In infrastructure mode, the wireless network contains at least one wireless access point (AP), a device that bridges wireless stations to each other and to a wired network. Stations that are members of a BSS are termed as “associated”. Stations cannot communicate directly with each other unless they go through the access point. The infrastructure mode is also referred as Basic Service Set (BSS) (Figure 2.3). Two

or more basic service sets connected by a distribution system is called an extended service set (ESS). In this research, we will focus on only infrastructure networks.

In ad-hoc mode, the wireless network contains no wireless APs. Wireless stations connect and communicate directly with each other. The ad-hoc mode is also known as Independent Basic Service Set (IBSS).

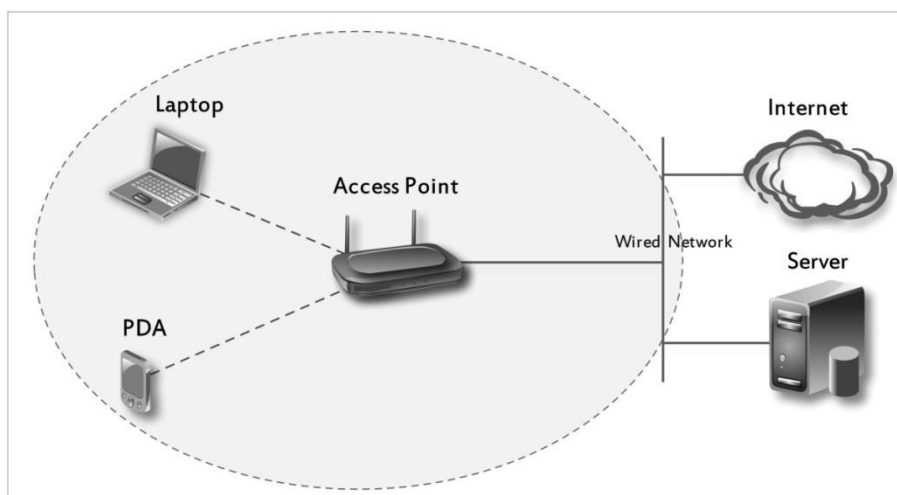


Figure 2.3 The infrastructure network (BSS).

The following sections will explain all stages of Figure 2.1 and Figure 2.2 in details.

2.1 Stage 1: Discovery of Security Capabilities

Within a BSS, prior to authentication to occur, a station and an access point (AP) should learn the RSN capabilities of each other. RSN security can be identified by a RSN information element (RSNIE) field found in certain 802.11 management frames. The RSN information element identifies the supported encryption cipher suites (WEP, TKIP, CCMP/AES) and the supported authentication methods (802.1X/EAP or PSK) of both the AP and the station (Figure 2.4). The RSN information element field is found in four different 802.11 management frames: beacon frames, probe response frames, association request frames and reassociation request frames.

| Element ID | Length | Version | Group Cipher Suite | Pairwise Cipher Suite Count | Pairwise Cipher Suite List | AKM Suite Count | AKM Suite List | RSN Capability | PMKID Count | PMKID List |
|------------|--------|---------|--------------------|-----------------------------|----------------------------|-----------------|----------------|----------------|-------------|------------|
|------------|--------|---------|--------------------|-----------------------------|----------------------------|-----------------|----------------|----------------|-------------|------------|

Figure 2.4 The RSN information element (IEEE, 2007).

The station discovers an access point by either active or passive scanning. *In passive scanning*, the station listens for the beacon frames that are continuously being sent by the access points (Figure 2.5). *In active scanning*, the station transmits probe requests to the AP which in turn replies with probe response (Figure 2.6). If the station hears beacons or receives probe responses from multiple access points, it will connect to the AP which has the best signal strength and quality.

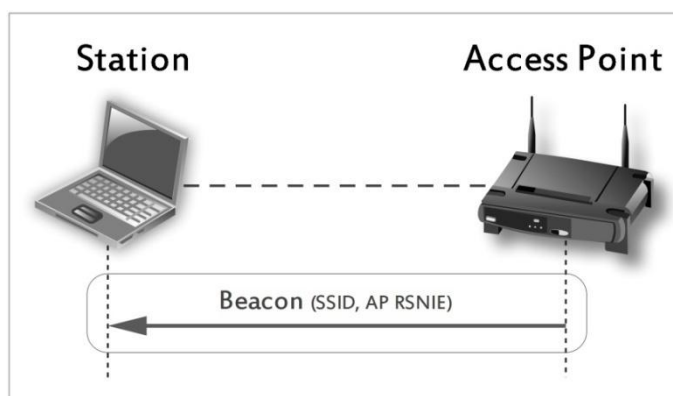


Figure 2.5 Passive scanning (IEEE, 2007).

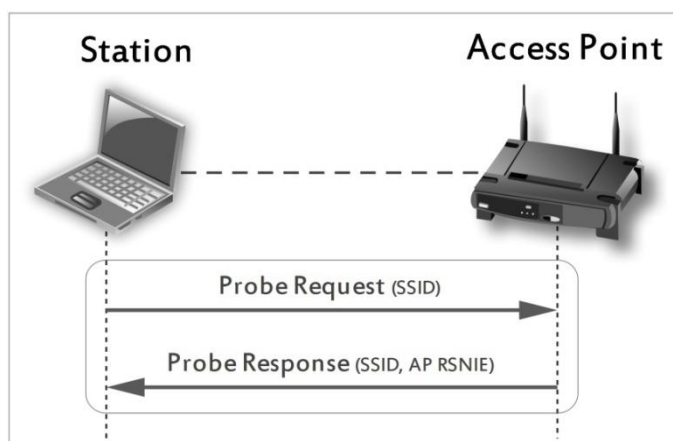


Figure 2.6 Active scanning (IEEE, 2007).

The access point learns about the station's security capabilities through association request frames or reassociation request frames sent by the station (Figure 2.7).

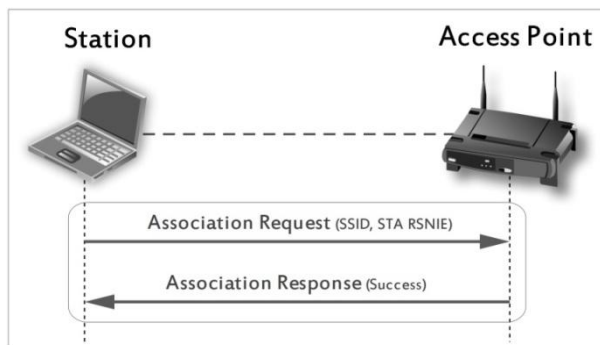


Figure 2.7 The 802.11 association (IEEE, 2007).

2.2 Stages 2 and 3: 802.11 Authentication and Association

The authentication and association states in WLAN are often misunderstood. Authentication is the first of two steps required to connect to the 802.11 network. Here, authentication doesn't mean to enter username and passwords in order to get access to the network resources. This authentication occurs at Layer 2 of the OSI model to create an initial connection between two stations. After the station has authenticated with the access point, the association process takes place. Once authentication and association occurs, the client STA establishes a Layer 2 connection to the AP and is considered as a member of the BSS. Only the associated station can send data through the access point to another device on the network. Both authentication and association must occur, in that order. Figure 2.8 shows the authentication and association states.

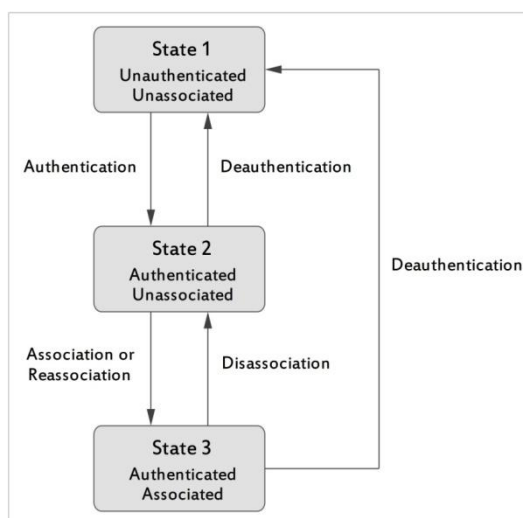


Figure 2.8 The 802.11 authentication and association states (Coleman & Westcott, 2009).

2.2.1 802.11 Authentication Methods

Open System authentication is considered as a null authentication. Every station is validated during Open System authentication, because there is no exchange or verification of identity between the devices. To provide data privacy, WEP encryption can be used only after authentication and association occur. Although Open System authentication does not provide any identity verifications, it is still used prior to the 802.1X/EAP authentication (Coleman, Westcott, Harkins & Jackman, 2010). It is the only pre-RSNA security mechanism that has not been deprecated. Open system authentication is a two-way authentication frame exchange, as shown in Figure 2.9.

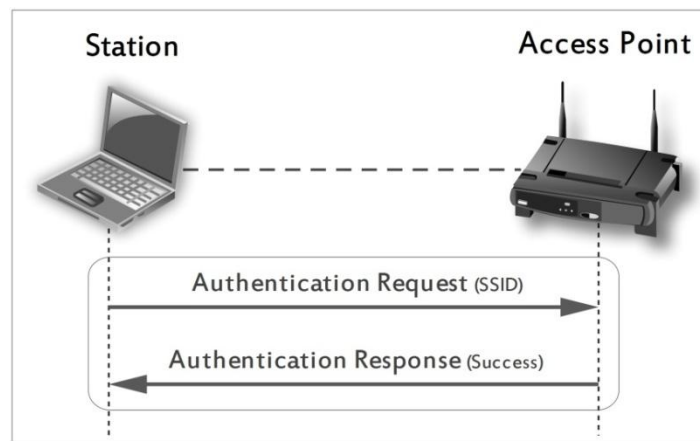


Figure 2.9 The 802.11 open system authentication (IEEE, 2007).

Shared Key authentication uses WEP keys to authenticate stations. The same static WEP keys must be manually configured on the AP and on all stations i.e. members of the BSS. Authentication will not work if the static WEP keys do not match. The same static WEP key that was used during the Shared Key authentication process will also be used to encrypt the 802.11 data frames (Figure 2.10).

2.2.1.1 Open System Authentication vs. Shared Key Authentication

It might seem Shared Key authentication is more secure than Open System authentication, since the Open System authentication offers no real authentication.

However, it is quite the opposite. With Open System authentication, anyone can associate to the access point but they can't pass traffic because they don't have the WEP key. When using Shared Key authentication, it is possible to derive the key stream used for the handshake by capturing the challenge frames. Hence, using Open System authentication together with WEP encryption is better than Shared Key authentication with WEP encryption (Figure 2.11) (Coleman, Westcott, Harkins & Jackman, 2010).

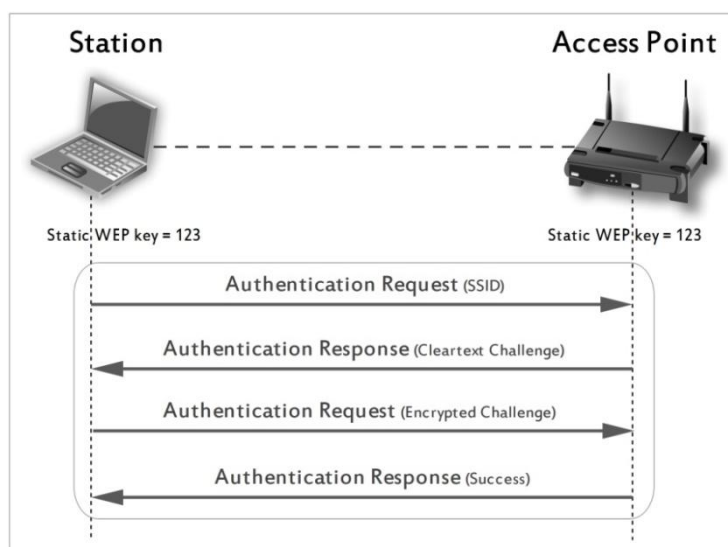


Figure 2.10 The 802.11 shared key authentication (Coleman, Westcott, Harkins & Jackman, 2010).

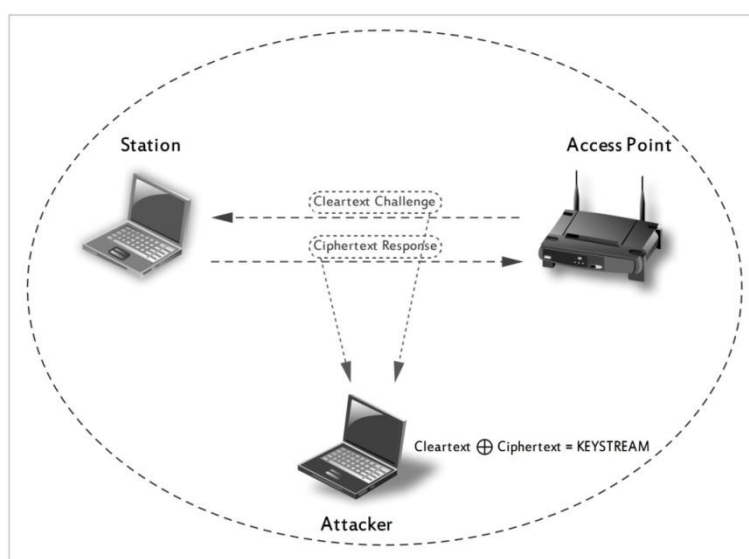


Figure 2.11 The vulnerability of shared key authentication.

2.2.2 802.11 Association

As it is said earlier, association occurs only after authentication. Associated station means that the station is a member of a basic service set (BSS) and it can send data through the access point. Association is also simple process that is done by two-way frame exchange as shown in Figure 2.7.

2.3 Stage 4: An 802.1X/EAP Authentication

The IEEE 802.11-2007 WLAN standard defines how 802.1X mechanisms are used for authentication and port control within an 802.11 WLAN. These mechanisms will be described in detail in this section. Before getting into the details of 802.1X/EAP Authorization framework, we should be sure about the following security concepts:

- **Authentication** is the verification of users' identity and credentials.
- **Authorization** is the allowing authenticated users to access to network resources and services. As it is clear, authentication occurs before authorization.

2.3.1 The 802.1X Standard

An IEEE 802.1X-2004 is a port-based access control standard that defines the mechanisms necessary to authenticate and authorize devices to use network resources. The 802.1X standard does not specify all of the components needed to implement a complete port-based authentication system, but it requires the use of several other standards and protocols, written by different organizations, such as an Extensible Authentication Protocol (EAP) and Remote Authentication Dial-in User Service (RADIUS) protocol. All of these standards and protocols work together and enable an 802.1X port-based authentication system to operate. The 802.1X operates at Layer 2 of OSI model with virtual ports of access points in WLAN. Every station within BSS is associated with the access point through virtual ports (IEEE, 2004a).

The 802.1X/EAP Authorization Framework consists of three main components:

Supplicant: A software application that performs the 802.1X endpoint services on a client device such as a laptop or PDA. There are many different types of supplicant client utility software exist (Figure 2.12). Each has its advantages and drawbacks. Some of them are free while some come with cost. Generally costly ones offer a more robust set of configuration parameters and can operate on multiple OS platforms and device platforms. When choosing supplicants the very important property is: its support for EAP-method type that is used within 802.1X/EAP authentication. Each supplicant has unique authentication credentials that are verified by the authentication server.

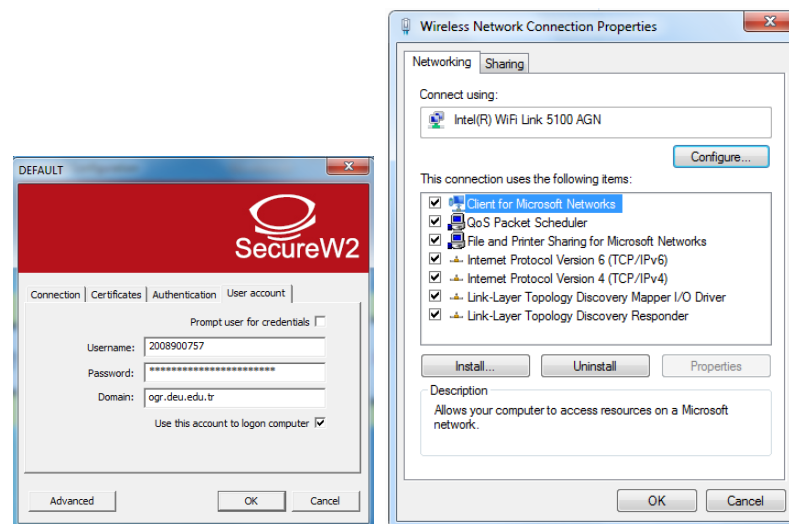


Figure 2.12 Windows 7 supplicant (right) and a secureW2 (left) which is the open-source EAP-TTLS client for Microsoft Windows platforms.

Depending on which EAP-method type is used, the supplicant identity credentials can be in many different forms as follows:

- Usernames and passwords
- Preshared keys (PSK)
- Digital certificates
- Smart cards
- Token devices
- RFID tags
- Biometrics

Authenticator: A Layer 2 device that blocks or allows traffic to pass through its port entity. It maintains two virtual ports: an uncontrolled port and a controlled port. The uncontrolled port allows only EAP authentication traffic to pass through, while the controlled port stays closed until the authentication server verifies the credentials of the supplicant. The authenticator does not validate the supplicant's credentials, it is essentially an intermediary device that passes certain messages between the supplicant and the authentication server. It is also important to understand that the authenticator doesn't need to know any specific EAP-method type, but just requires EAP authentication. In a WLAN, the authenticator is usually either an Access Point or a WLAN controller.

Authentication Server: A server that validates the credentials of the supplicant that is requesting access. The authentication server and the supplicant communicate using a Layer 2 EAP authentication protocol. If the supplicant's credentials are successfully verified, the authentication server notifies the authenticator that the supplicant has been authorized. The Table 2.1 contains several examples of authentication servers. Typically a Remote Authentication Dial-in User Service (RADIUS) server is used as an authentication server. But any Lightweight Directory Access Protocol (LDAP) - compliant database can be used as the authentication server, too.

Table 2.1 Widely deployed authentication servers (Coleman, Westcott, Harkins & Jackman, 2010).

| Product Name | Protocol |
|--|-------------------|
| Cisco ACS | RADIUS |
| Juniper Steel Belted RADIUS | RADIUS |
| Microsoft NAP (Windows Server 2008) | RADIUS |
| Microsoft AD 2003 and higher | Kerberos and LDAP |
| FreeRADIUS (open source) | RADIUS |

The authentication server will maintain a user database or may proxy with an external user database to authenticate user credentials (Figure 2.13). In some cases, the authentication server may be embedded in the authenticators. This authentication server model significantly reduces authentication traffic over the network, thus increases authentication performance. This can be particularly useful in small sites.

The embedded authentication servers that are incorporated into many of the WLAN APs and controllers are not as full featured as the dedicated RADIUS servers.

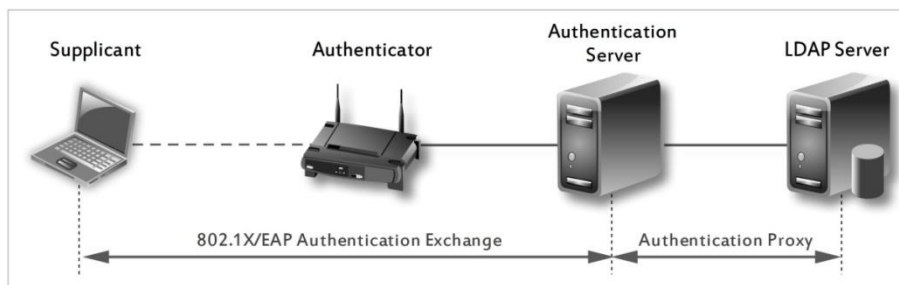


Figure 2.13 Proxy authentication (Coleman, Westcott, Harkins & Jackman, 2010).

In order to communicate with each other, the RADIUS server and the authenticator need to be configured with each others' IP Addresses, UDP ports (1645 or 1812) and with a *shared secret*. The shared secret is only used to validate and encrypt the communication link between the authenticator and the server.

Not only the supplicant, but also the authentication server needs to present its credentials to the supplicant when there is mutual authentication. Strong EAP authentication methods provide mutual authentication between the supplicant and the server to prevent primarily man-in-the-middle attacks and other such attacks.

2.3.2 Extensible Authentication Protocol (EAP)

The Extensible Authentication Protocol (EAP) is the Layer 2 protocol used within an 802.1X framework. EAP is designed flexible to support many different specific authentication protocols. EAP is a lock-step protocol, which means only one packet is delivered at a time in order, out of order reception is not supported. In other words, other than the initial Request, a new Request cannot be sent prior to receiving a valid response. The 802.1X components are referred to as the followings:

- Supplicant : Peer
- Authenticator : Network Access Server (NAS)
- Authentication server : EAP server/AAA server

NAS devices need to support the 802.1X in order to use EAP, but they do not have to understand each authentication method and may act as a pass-through agent for a backend authentication server (Figure 2.14) (Aboba, Blunk, Vollbrecht, Carlson & Levkowitz, 2004).

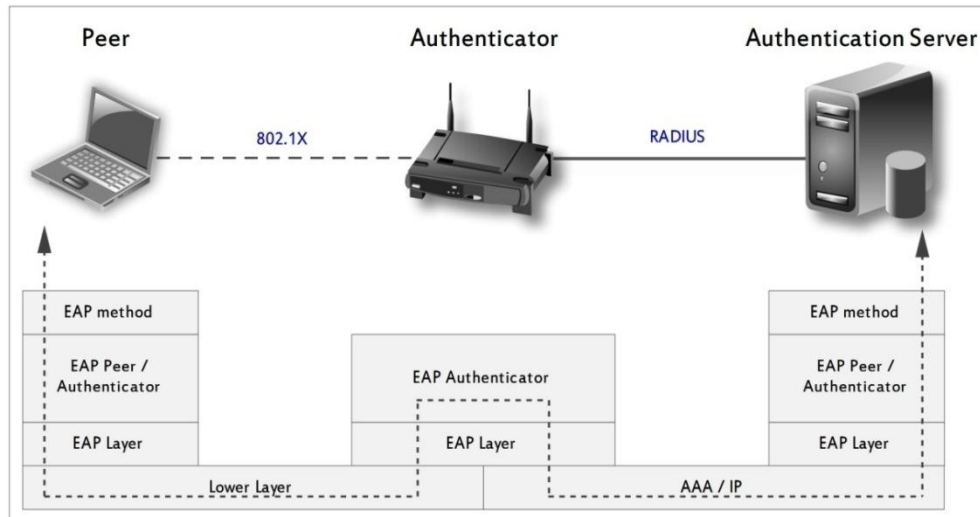


Figure 2.14 Pass-through mode of authenticator (Aboba, Blunk, Vollbrecht, Carlson & Levkowitz, 2004).

There are four EAP frame types: *Request*, *Response*, *Success* and *Failure*. The supplicant can only issue EAP-Response frames, and the authenticator can perform EAP-Request, Success, and Failure frames. EAP-Request and EAP-Response packets carry the specific EAP-Method protocol data while EAP-Success and EAP-Failure packets carry no data but the result of the authentication process.

2.3.3 EAP Carrier Protocols

2.3.3.1 EAPOL Protocol

A specific EAP-method protocol implements the actual authentication process between a supplicant and an authentication server. EAP packets carry the EAP-Method protocol data. EAPOL packets transport the EAP packets, and 802.11 data frames carry the EAPOL packets between the supplicant and the authenticator. The encapsulation of packets is shown in Figure 2.15.

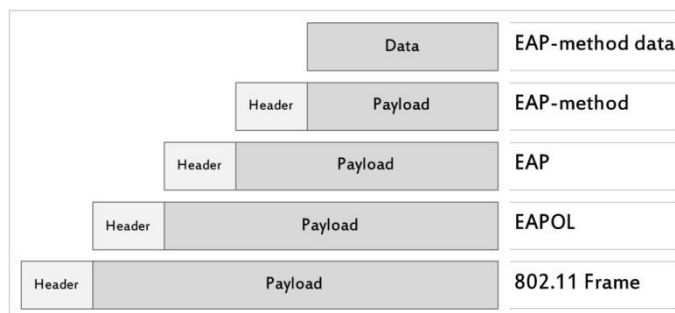


Figure 2.15 EAPOL encapsulation (Geier, 2008).

There are five major types of EAPOL messages, from which only one type carries EAP packets. Table 2.2 summarizes the EAPOL frames.

Table 2.2 EAPOL packets (Coleman, Westcott, Harkins & Jackman, 2010).

| Name | Description |
|-------------------------------------|--|
| EAP-Packet | Only this frame carries EAP packets. |
| EAPOL-Start | The supplicant can use this frame to initiate the EAP process. This frame is optional. |
| PEAPOL-Logoff | This frame terminates an EAP session and return the authenticated port to an unauthorized state. |
| EAPOL-Key | This frame is used to exchange dynamic keying information. |
| EAPOL-Encapsulated-ASF-Alert | This frame is used to send alerts. |

The EAPOL-Start and the EAPOL-Encapsulated-ASF-Alert frames are only sent by the supplicant to authenticator, while other frames can be sent to each side.

2.3.3.2 RADIUS Protocol

RADIUS provides the “*transportation*” of the EAP packets between the authenticator and the authentication server. RADIUS frames are sent using a lock-step mechanism i.e. frames are sent in order. All EAP-method data is transported in encrypted format (Aboba & Calhoun, 2003). The followings are RADIUS frame types:

- Access-Request
- Access-Challenge
- Access-Accept
- Access-Reject
- Accounting-Request
- Accounting-Response

An Access-Request and an Accounting-Request frames are sent by the authenticator to the RADIUS server. The other frames are sent by RADIUS server to the authenticator. As it is seen, the authenticator acts as a translator between the supplicant and the RADIUS server (Figure 2.16).

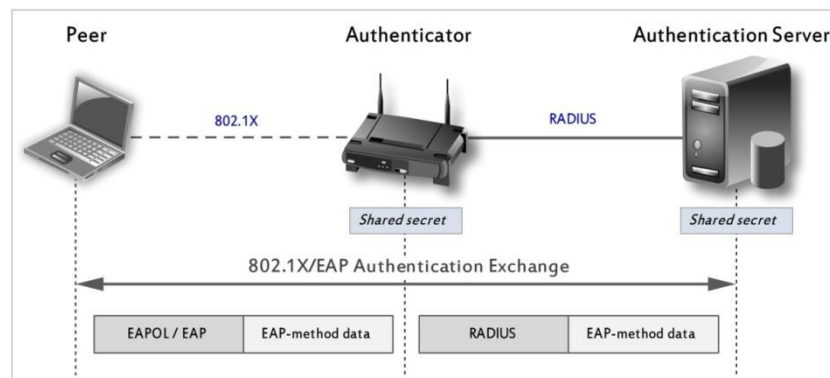


Figure 2.16 Authenticator acts as a translator (Geier, 2008).

2.3.4 EAP Methods

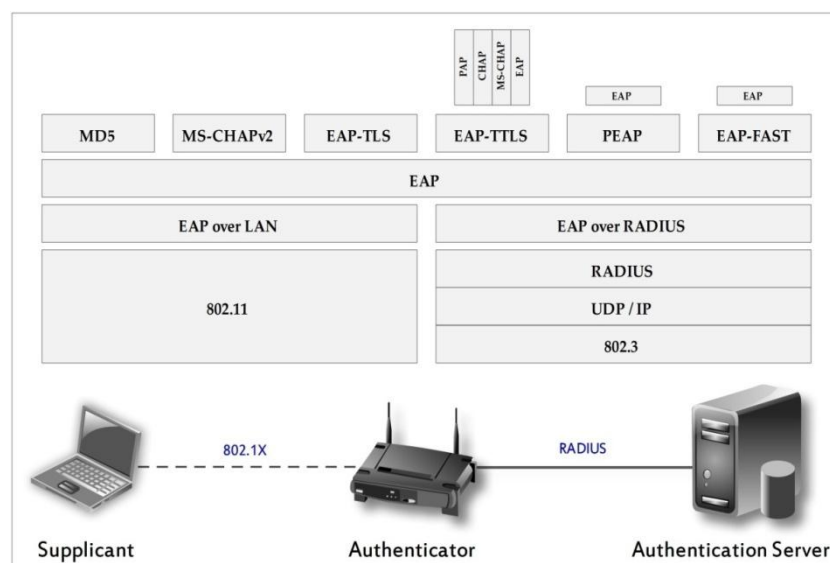


Figure 2.17 An 802.1X layering (Haas, 2010).

An EAP-Method actually implements the authentication process, whereas other protocols, such as EAPOL and RADIUS, merely transport the EAP-Method data. The layered authentication framework is shown in Figure 2.17 and the generic EAP exchange is shown in Figure 2.18.

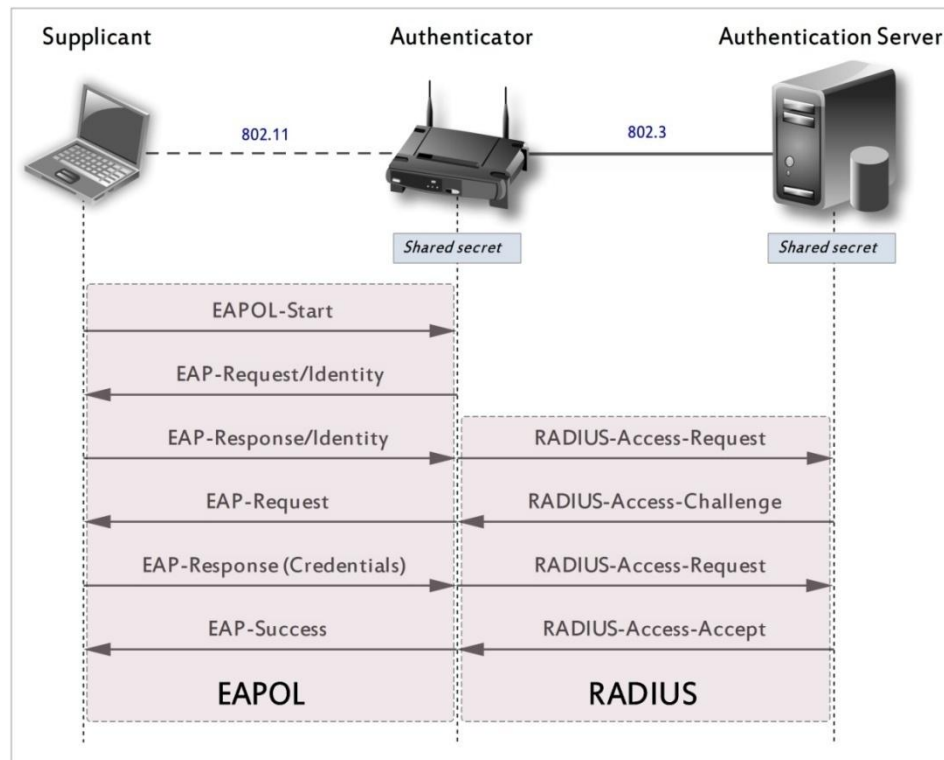


Figure 2.18 Generic EAP exchange (Coleman, Westcott, Harkins & Jackman, 2010).

There are a number of EAP-methods, some are defined in RFCs and many others are proprietary. EAP-Methods make use of different types of credentials, such as username/passwords, pre-shared keys and digital certificates. The EAP specification, RFC 3748, defines three EAP-methods. They are EAP-MD5 (MD5 Challenge), EAP-OTP (One-Time Passwords) and EAP-GTC (Generic Token Card). These EAP-methods are very simple and provide only one-way authentication, thus there is no generation of keys. These methods do not meet requirements of EAP protocol, they should be avoided. All EAP implementations are required to support these methods. As weak EAP-methods do exist, there are also very strong EAP-methods. In next chapter we will discuss in details the TLS-based EAP methods, which are very popular and widely used in today's networks.

2.4 Stages 5, 6 and 7: Key Management

The IEEE 802.11-2007 WLAN standard defines how 802.11 and 802.1X mechanisms are used together to provide for robust secure key management. This section focuses on key management procedures.

The goals of authentication and encryption are very different. Authentication provides mechanisms for verification of users' identity and credentials, while encryption provides mechanisms for data privacy or confidentiality. But they are linked together in AKM services. The authentication process provides the seeding material to create the necessary encryption keys i.e. encryption keys cannot be created without authentication.

2.4.1 RSNA Key Hierarchy

A successful 802.1X/EAP mutual authentication will generate a key known as Master Session Key. Both, the supplicant and the authentication server will create the same MSK separately. The generation of the MSK from the EAP process is EAP method specific. The MSK is also referred to as the AAA key.

The MSK is used as seeding material to create another master key called Pairwise Master Key (PMK). The PMK is simply computed as the first 256 bits (bits 0–255) of the MSK. The PMK derivation will occur in both parties: the supplicant and the authentication server. Every supplicant will have its own unique PMK. PMK is generated every time the supplicant authenticates or reauthenticates (IEEE, 2007). After the generation of the PMK, the authentication server securely transfers the PMK to authenticator (Figure 2.1, Stage 5). The server will delete the PMK from its disk.

In SOHO environments, where there is no 802.1X/EAP solution, preshared key becomes the PMK (Figure 2.2, Stage 4). In fact, SOHO users are more familiar with using passwords rather than preshared keys. In this case, preshared key can be

generated from password. The formula to convert a password to a PSK is given below:

$$\text{PSK} = \text{PBKDF2} (\text{passphrase}, \text{ssid}, \text{ssidLength}, 4096, 256)$$

where

PSK : preshared key,

PBKDF2 : password-based key generation function,

passphrase : user password,

ssid : an 802.11 wireless network name,

ssidLength : the number of octets of the ssid,

4096 : the number of times the passphrase is hashed,

256 : the number of bits output by the passphrase mapping.

The above PSK generation process will occur in each station that is a member of BSS in SOHO environment. As a result, every station will have the same PMK. It should be noted that, weak passwords are highly susceptible to social engineering attacks and offline dictionary attacks (Coleman, Westcott, Harkins & Jackman, 2010).

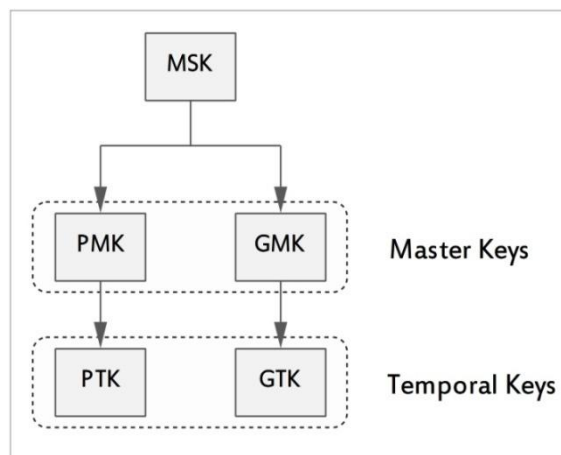


Figure 2.19 Key hierarchy of RSN.

Another master key, known as the group master key (GMK), is randomly created on the authenticator. The PMK and the GMK master keys are not used to encrypt or decrypt 802.11 data. They will be used as seeding material for the Four-Way Handshake process which creates temporal keys that are used to encrypt and decrypt

802.11 data frames between the station and the access point. The keys generated from the Four-Way Handshake are called the pairwise transient key (PTK) and the group temporal key (GTK). The PTK is generated using the PMK and the GTK is generated using GMK keys (Figure 2.19) (IEEE, 2007).

The PTK is unique between each individual station and the access point and it encrypts all unicast transmissions between them. PTK is composed of three sub keys:

- *Key Confirmation Key (KCK)* is used to provide data integrity during the 4-Way Handshake and Group Key Handshake.
- *Key Encryption Key (KEK)* is used by the EAPOL-Key frames to provide data privacy during the 4-Way Handshake and Group Key Handshake.
- *Temporal Key (TK)* is used to encrypt and decrypt the 802.11 data frames between the supplicant and the authenticator (Figure 2.20).

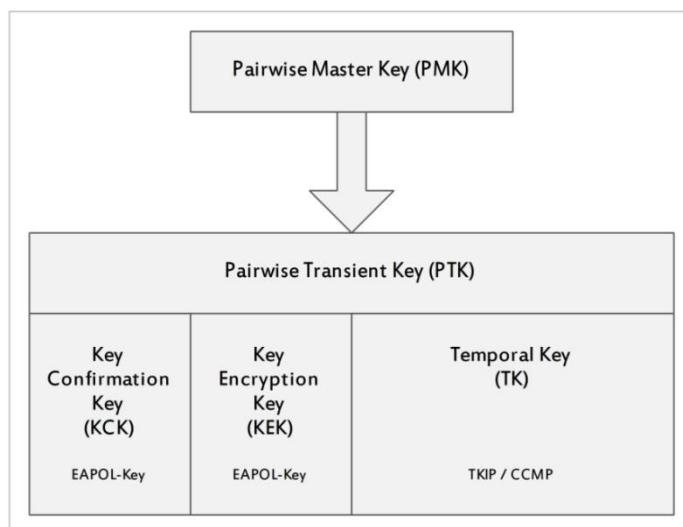


Figure 2.20 Pairwise transient key (Coleman, Westcott, Harkins & Jackman, 2010).

The GTK is shared among all stations and the single access point. GTK is used to encrypt all broadcast and multicast frames (Figure 2.21).

The PTKs and the GTKs used for encryption are either CCMP/AES or TKIP/RC4.

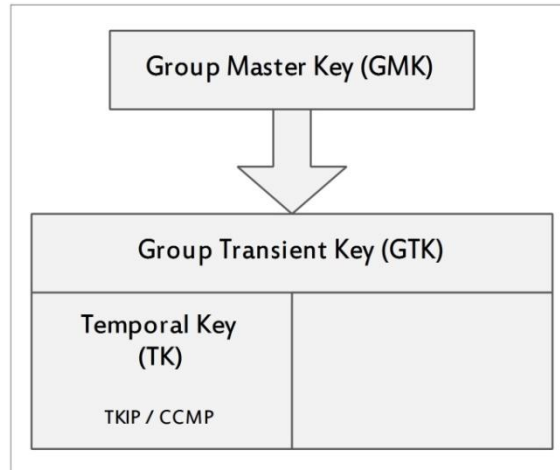


Figure 2.21 Group temporal key (Coleman, Westcott, Harkins & Jackman, 2010).

2.4.2 The Four-Way Handshake

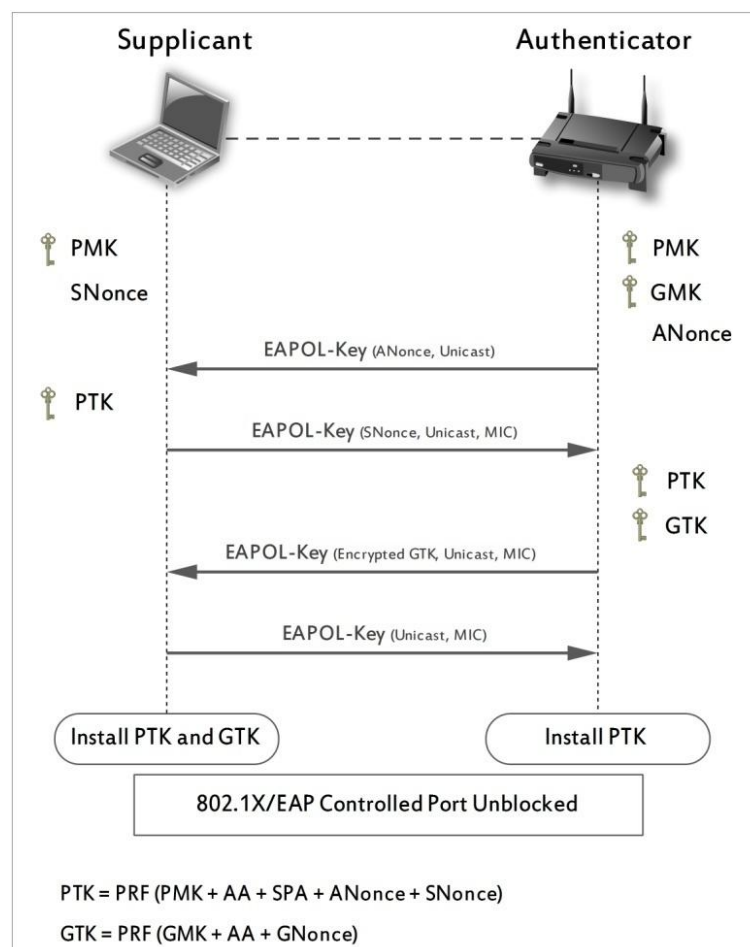


Figure 2.22 The four-way handshake protocol (IEEE, 2007).

As mentioned above, the Four-Way Handshake finalizes the AKM process by generating PTK for encryption of unicast transmissions and a GTK for encryption of broadcast/multicast transmissions. The Four-Way Handshake process occurs between the supplicant and the authenticator. The EAPOL-Key frame messages are used within Four-Way Handshake process to confirm the existence of the same PMK, verify the selection of the cipher suite, derive and install a fresh PTK for the following data session. The authenticator might also distribute a GTK to the supplicant if necessary. After the successful Four-Way Handshake, the virtual controlled port of the authenticator is unblocked. All 802.11 data frames that are encrypted with appropriate keys are can pass through the authenticator (Figure 2.22). The complete message exchange details of the Four-Way Handshake process are given in chapter four in Alice & Bob Notation form.

2.4.3 The Group Key Handshake

An authenticator may change the GTK on disassociation or deauthentication of a client station. In such cases, the authenticator will generate a fresh Group Transient Key (GTK) and distribute this GTK to the supplicants. The Group Key Handshake is used only to issue a new GTK to all stations that already have an original GTK generated by an earlier Four-Way Handshake. The Group Key Handshake is identical to the last two frames of the Four-Way Handshake process (Figure 2.23).

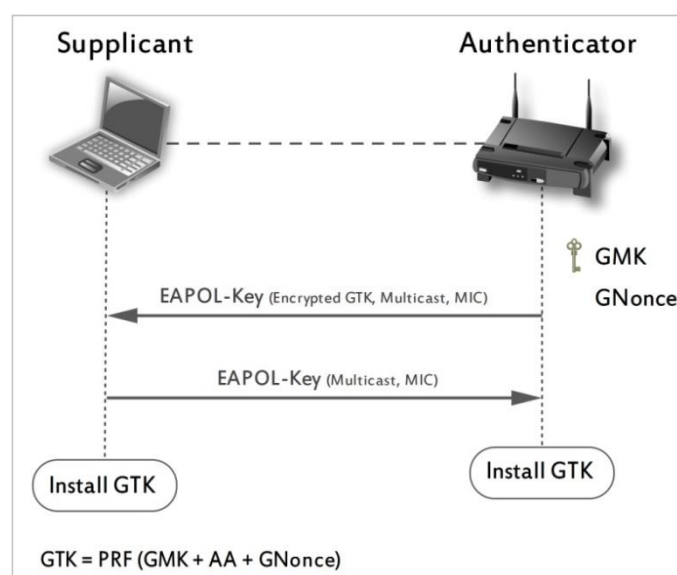


Figure 2.23 The group key handshake protocol (IEEE, 2007).

2.5 Stage 8: Secure Data Communication

The 802.11-2007 standard defines three encryption methods that operate at Layer 2 of the OSI model: WEP, TKIP, and CCMP. All these encryption methods use symmetric algorithms. Symmetric algorithms are faster and require less computer processing power than asymmetric algorithms. Using the PTK (or GTK) and the negotiated cipher suite from above handshakes, all upper layer data, through layer 3 to layer 7, is encrypted prior to transmission and then decrypted after being received. The PTKs and the GTKs used for encryption may be either TKIP/RC4 or CCMP/AES (Coleman, Westcott, Harkins & Jackman, 2010) (Figure 2.24).

Wired Equivalent Privacy (WEP) is a Layer 2 security protocol that uses the RC4 streaming cipher. WEP uses a preconfigured static key that is shared between access point and all stations. WEP runs a cyclic redundancy check (CRC) for data integrity. It is not cryptographically strong integrity protection. Due to its many vulnerabilities, WEP has been deprecated. WEP is still supported only for backward compatibility within TSN.

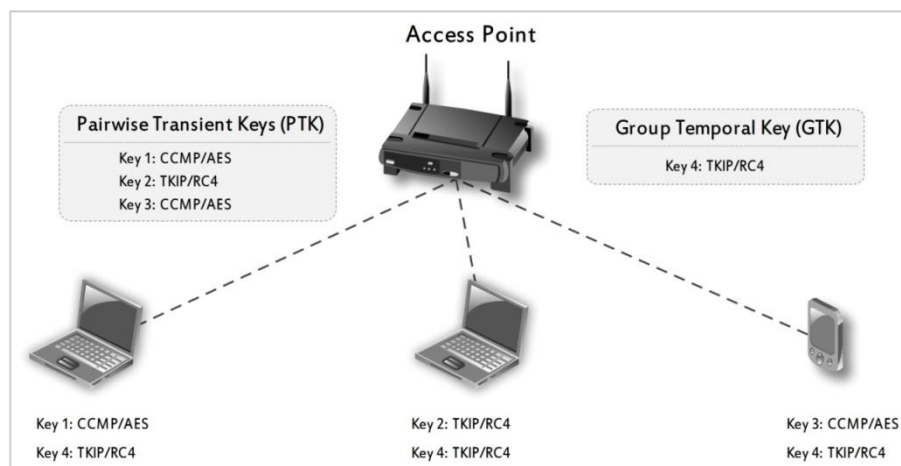


Figure 2.24 RSNA within BSS.

Temporal Key Integrity Protocol (TKIP) is an enhancement of WEP that also uses the RC4 algorithm as WEP does. It was created to provide a stronger security solution without requiring users to replace their legacy equipment. With just a firmware upgrade, it is possible to use TKIP within legacy equipments. TKIP uses dynamically created encryption keys and a stronger data integrity check known as

the Message Integrity Code (MIC). TKIP addresses many known weaknesses of WEP. They are *social engineering attacks, replay attacks, reinjection attacks, weak key attacks, bit-flipping attacks, forgery attacks, impersonation attacks, fragmentation attacks*.

TKIP was a short term solution. TKIP has been successfully used for five years until when some flaws were found in TKIP such as Beck-Tews attack, Ohiagi/Morii attack (Coleman, Westcott, Harkins & Jackman, 2010).

Counter Mode with Cipher-Block Chaining Message Authentication Code Protocol (CCMP) was designed to replace TKIP and WEP. CCMP uses the AES block cipher algorithm. Legacy 802.11 devices that only supported WEP and TKIP had to be replaced with newer hardware to support CCMP/AES encryption processing. CCMP is made up of many different components that provide different functions. The Counter Mode (CTR) is used to provide data confidentiality. The Cipher-Block Chaining Message Authentication Code (CBC-MAC) is used for authentication and integrity. CCMP is mandatory in WPA2 networks, while TKIP/RC4 is mandatory in WPA networks (Coleman, Westcott, Harkins & Jackman, 2010). The Table 2.3 depicts the properties of encryption methods used in 802.11.

Table 2.3 The 802.11 encryption methods.

| Encryption method | Cipher | Key Generation | Integrity | Comments |
|--|--------|----------------|-----------|--|
| WEP <i>(Wired Equivalent Privacy)</i> | RC4 | Static | ICV (CRC) | <ul style="list-style-type: none"> • Has weaknesses • Has been cracked • Still deployed in enterprise |
| TKIP <i>(Temporal Key Integrity Protocol)</i> | RC4 | Dynamic | MIC | <ul style="list-style-type: none"> • Enhancement of WEP • Needs firmware upgrade • Has flaws |
| CCMP <i>(Counter Mode with Cipher-Block Chaining Message Authentication Code Protocol)</i> | AES | Dynamic | CBC-MAC | <ul style="list-style-type: none"> • Processor intensive |

CHAPTER THREE

TLS-BASED EAP METHODS

The 802.1X does not specify an exact authentication method. The 802.1X uses the concept of an EAP framework that allows a variety of specific methods to be used for the authentication procedure. An EAP-method actually implements the authentication process between a peer and an authentication server. There are two major sets of EAP-methods, which are password-based and certificate-based. The password-based EAP-types provide lightweight processing and are very convenient. But many of them are susceptible to the offline dictionary attacks, and hence considered weak. On the other hand, the certificate-based methods provide strong security as well as allow password-based authentication methods to be used. The certificate-based methods achieve these security properties using Transport Layer Security (TLS) Handshake protocol that establishes authenticated and encrypted tunnel (Dierks & Rescorla, 2006, 2008). Within tunnel, password-based methods can run securely. The significant downside of certificate-based methods is the requirement of Public Key Infrastructure (PKI) which is costly to implement and hard to manage. As a result, there is a need for an EAP method that can provide the same level of security as certificate-based types as well as allows password-based methods run on it. EAP-FAST is the exactly protocol that we need. The EAP-FAST does not use certificates, instead it uses shared secret within TLS handshake protocol to establish secure tunnel and it does allow any password-based EAP-methods run within the tunnel.

3.1 TLS-Based EAP Methods Overview

In this chapter, we will discuss and compare the security properties of the widely used TLS-based EAP-methods which are defined in IETF RFCs used in WLAN (Table 3.1). Table 3.2 lists the EAP types that currently included in the Wi-Fi Alliance Certification program. All these EAP-methods use a TLS Handshake protocol. EAP-TTLS, PEAP and EAP-FAST methods are tunnel-based methods that extend the EAP-TLS protocol. Tunnel-based methods are constructed as combination

of two protocols: an outer protocol and an inner protocol. The outer protocol is the TLS Handshake protocol which establishes encrypted TLS tunnel to protect the exchange of the inner protocol messages. The inner protocol is usually the weak password-based protocol. Weak, legacy protocols are used as an inner protocol because they are already widely deployed and work lightweight. The tunnel-based protocols provide mutual authentication and run in two phases. In the first phase, the outer protocol runs and authenticates the server to the peer. The inner protocol is typically used for peer authentication, in the second phase. As a result of successful authentications, both the outer and the inner protocols derive some keys (Figure 3.1). Among TLS-based protocols, in this chapter we mainly focus on the EAP-FAST protocol because of its attracting security features.

Table 3.1 TLS-based EAP methods defined in IETF RFCs.

| EAP-types | RFCs | Category | Publication Date |
|------------|---|-----------------|------------------|
| EAP-TLS | RFC 5216 | Standards Track | March, 2008 |
| EAP-TTLSv0 | RFC 5281 | Informational | August, 2008 |
| EAP-TTLSv1 | draft-funk-eap-ttls-v1-01.txt | Informational | March, 2006 |
| PEAPv0 | draft-kamath-pppext-peapv0-00.txt | Informational | October, 2002 |
| PEAPv1 | draft-josefsson-pppext-eap-tls-eap-05.txt | Informational | September, 2002 |
| PEAPv2 | draft-josefsson-pppext-eap-tls-eap-10.txt | Informational | October, 2004 |
| EAP-FASTv1 | RFC 4851 | Informational | May, 2006 |
| EAP-FASTv2 | draft-ietf-emu-eap-tunnel-method-01.txt | Standards Track | October, 2011 |

Table 3.2 TLS-based EAP methods included in the WFA certification program

| EAP Types | Comments |
|---------------------|---|
| EAP-TLS | Client certificate can be stored on a smartcard |
| EAP-TTLS/MSCHAPv2 | Well supported by Cisco and Microsoft |
| PEAPv1/EAP-GTC | Not supported by Windows OS, so not really deployed |
| PEAPv0/EAP-MSCHAPv2 | Method mainly supported by Microsoft |
| EAP-FAST | A protocol proposal by Cisco Systems |

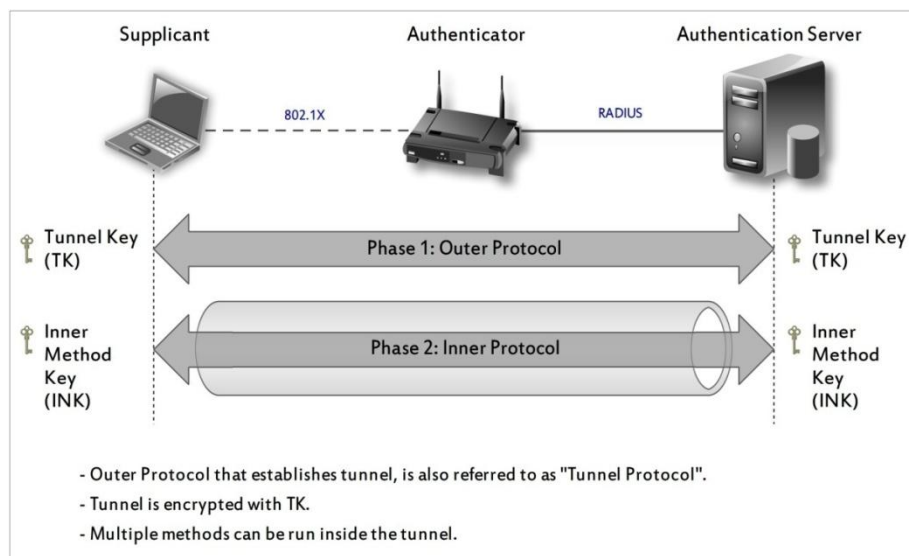


Figure 3.1 Tunnel-based EAP methods overview (Hoepfer & Chen, 2009).

Tunnel-based EAP methods were introduced for several reasons:

- To enable the use of password-based authentication methods for peers. As mentioned before, without tunneling, widely deployed password-based authentication methods are insecure.
- To enable privacy protection. Not only the peer identity but also the server identity can be protected.
- To enable the execution of multiple authentication methods. In cases, where both a machine authentication and the user authentication are required we will need to provide multiple authentications. Since a tunnel-based EAP method is considered as one authentication method and, thus, multiple authentication methods may be executed within the protective tunnel.

3.2 EAP-TLS

EAP-Transport Layer Security (EAP-TLS) is defined in RFC 5216 and is considered one of the most secure EAP methods available today. The EAP-TLS has the broadest support in supplicants and authentication servers. EAP-TLS requires both the peer and the authentication server have X.509 certificates for authentication. This means that each client requires a unique digital certificate. It is difficult to

manage the certificates in a large enterprise network, since certificates add administrative overhead. As a result, EAP-TLS is rarely deployed. EAP-TLS is best for enterprises that have digital certificates already deployed. Another drawback of EAP-TLS is that the peer identity is exchanged in the clear. So, a passive attack can easily obtain the usernames. EAP-TLS provides mutual authentication as shown in Figure 3.2 (Simon, Aboba & Hurst, 2008).

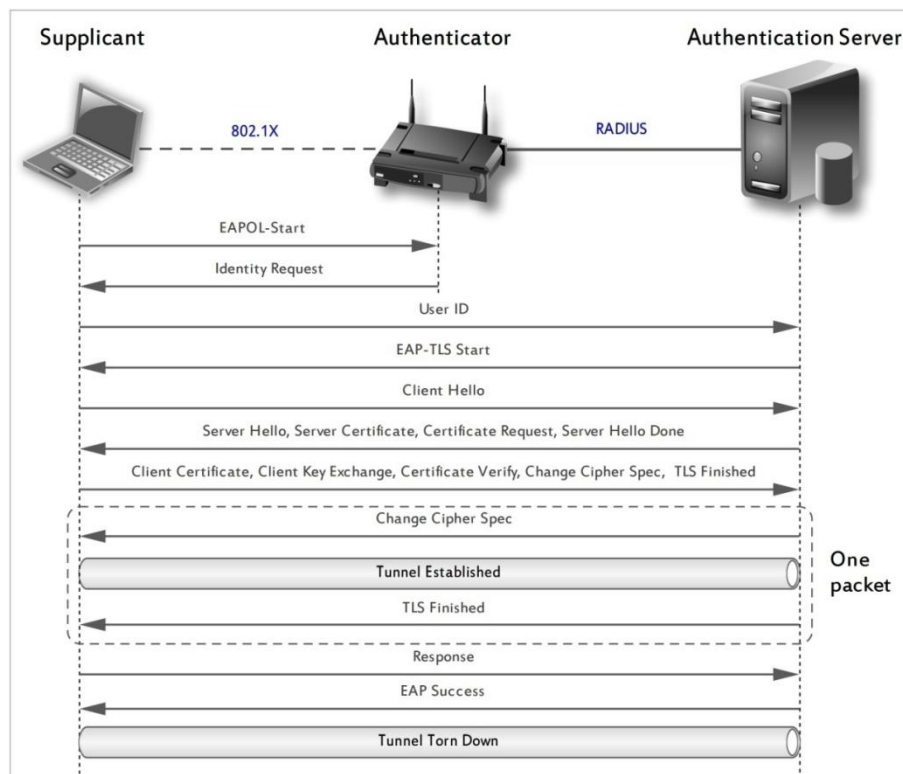


Figure 3.2 EAP-TLS authentication mechanism (Simon, Aboba & Hurst, 2008).

3.3 EAP-TTLS

EAP-Tunneled Transport Layer Security (EAP-TTLS) is a two-phase authentication protocol that establishes encrypted tunnel in phase one, and then performs user authentication within encrypted tunnel in phase two. The EAP-TTLS requires only server-side certificates for server authentication. The users can authenticate themselves to the server through the use of a password, rather than a certificate. This significantly reduces the complexity of the port-based authentication system. The EAP-TTLS supports both EAP protocols and non-EAP protocols such

as PAP, CHAP, MS-CHAPv1, MS-CHAPv2 within encrypted tunnel. TTLS uses the TLS tunnel to exchange "attribute-value pairs" (AVPs), much like RADIUS. Note that, in phase one the real user identity is hidden (Funk & Blake-Wilson, 2008).

3.4 PEAP

Protected Extensible Authentication Protocol (PEAP) is often called as "EAP inside EAP". PEAP is the most common and most widely supported EAP-method. PEAP operates in two phases similar to EAP-TTLS. PEAP also supports the identity hiding, as EAP-TTLS. Moreover, PEAP provides the chaining of several EAP-methods, cryptographic binding of outer and inner methods. These properties differentiates PEAP from EAP-TTLS. Figure 3.3 illustrates the PEAP authentication. Note that, EAP-TTLS is very similar to PEAP (Palekar & others, 2004).

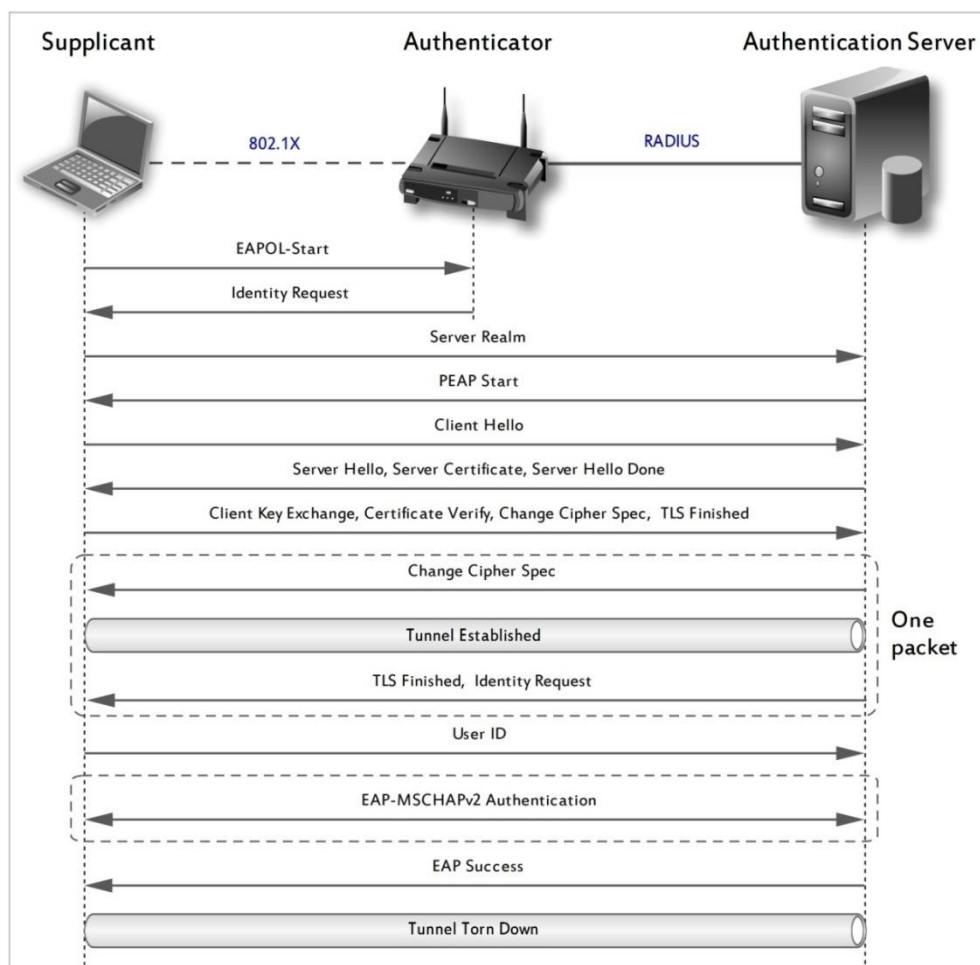


Figure 3.3 PEAP authentication mechanism (Palekar & others, 2004).

3.5 EAP-FAST

Flexible Authentication via Secure Tunneling EAP (EAP-FAST) is a "lightweight" and convenient protocol that can provide the same level security as PEAP and EAP-TTLS. Unlike PEAP and EAP-TTLS, EAP-FAST uses a Protected Access Credential (PAC) to establish a TLS tunnel instead of X.509 digital certificates. With using PACs, EAP-FAST authentication acts more like a session resumption, hence the authentication occurs much more faster than complete authentication. Use of server certificates is optional in EAP-FAST (Cam-Winget, McGrew, Salowey & Zhou, 2007).

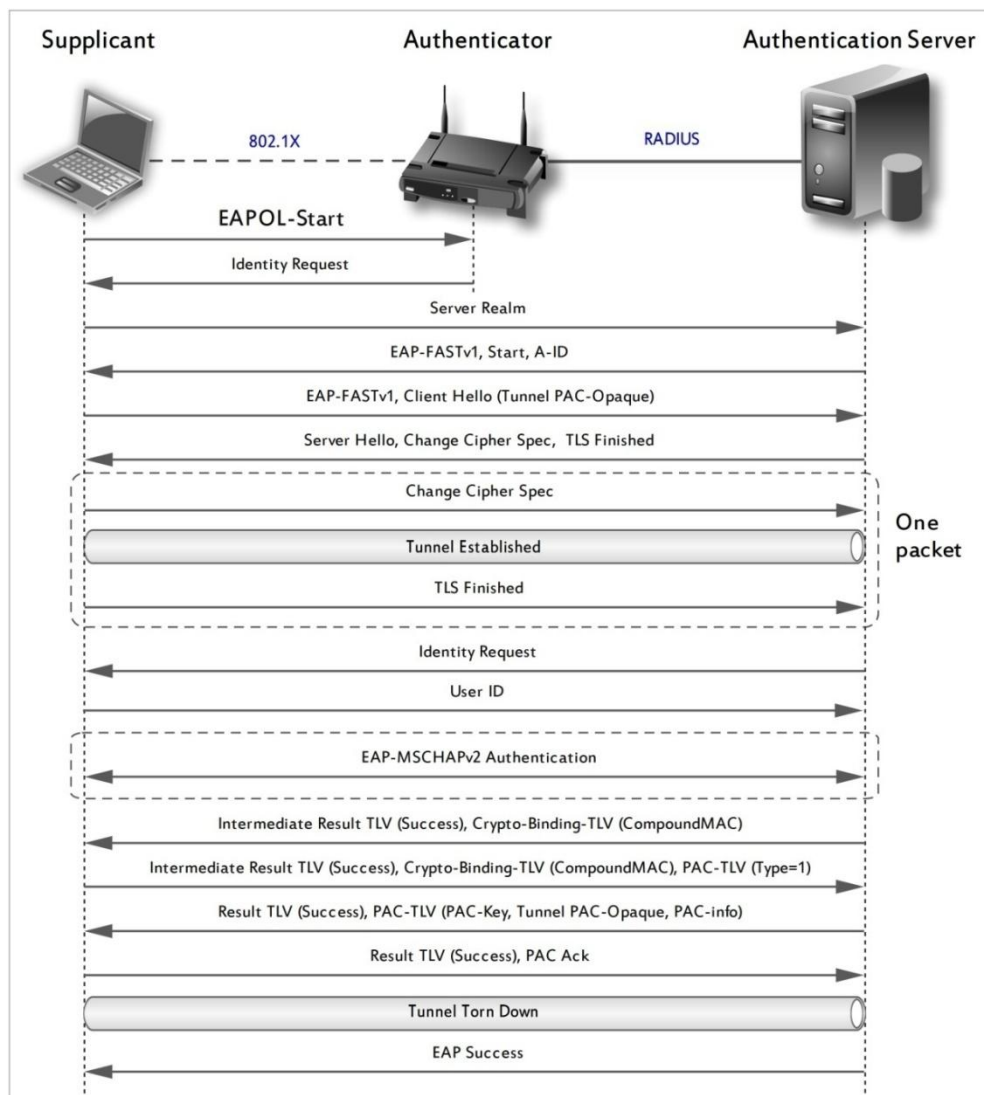


Figure 3.4 EAP-FAST authentication (Cam-Winget, McGrew, Salowey & Zhou, 2007).

EAP-FAST consists of three phases: Phase 0 is an optional phase in which the PAC can be provisioned manually or dynamically. This phase may be skipped in the case of the peer has appropriate PACs. PAC provisioning is only done once to set up the PAC secret between the server and client and all subsequent EAP-FAST sessions skip "Phase 0". Phase 0 is independent of other phases. In Phase 1, the client and the AAA server uses the PAC to establish TLS tunnel. In Phase 2, the client credentials are exchanged inside the encrypted tunnel. Figure 3.4 depicts the EAP-FAST process.

3.5.1 PAC Types

- *Tunnel PAC* is used to establish an authenticated and encrypted tunnel between the peer and the authentication server. The Tunnel PAC is consists of PAC-Key, PAC-Opaque and PAC-Info. PAC-Key is a shared secret key that will be used within generation of tunnel key. PAC-Opaque is the protected data that can not be interpreted by the peer. Only the authentication server can decrypt it. Figure 3.5 depicts the Tunnel PAC.

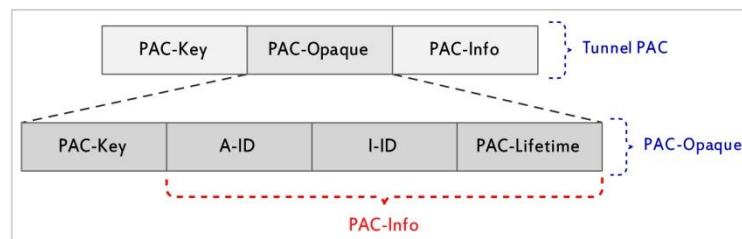


Figure 3.5 Tunnel PAC.

- *Machine Authentication PAC* contains PAC-Info that is used in identification of the machine. This PAC can be provisioned during the authentication of a user and can also be used in establishing a secure tunnel as the Tunnel PAC.
- *User Authorization PAC* is also PAC-Info that holds user identity information. When this PAC is presented in phase 2 of EAP-FAST, inner authentication process may be skipped.

3.5.2 Dynamic PAC Provisioning

As shown in Figure 3.5, the Tunnel PAC contains the PAC-Key, PAC-Opaque and PAC-Info. The PAC-Opaque contains the PAC-Key, initiator ID (I-ID) and the Key Lifetime. I-ID is assigned by the authentication server to the peer and it is only used by the server for peer identification. The PAC Info contains the Authenticator ID (A-ID) and A-Info, both of which identify the particular authentication server that created the PAC for the specific I-ID (Peer). All of these are created by the authentication server. The authentication server encrypts the PAC-Opaque with its own Master Key. Within encrypted tunnel, the authentication server sends the created Tunnel PAC to the peer. The authentication server deletes the Tunnel PAC from its memory to save the storage capacity.

After possessing the valid Tunnel PAC, the peer will reauthenticate to use PAC. The peer will skip the phase 0 and starts directly from phase 1. In phase 1 the authentication server will send its A-ID to the peer. The peer uses the A-ID to select the correct PAC from its inventory (it may have multiple PACs, one for each Server it may authenticate with). The peer sends the correct PAC-Opaque to the authentication server. The authentication server decrypts the PAC-Opaque using its Master Key (the same one that originally encrypted the PAC-Opaque) and obtains the PAC-Key. Now both parties, the peer and the authentication server holds the same key. Thus they use this key in generation a Tunnel Key. A secure tunnel is now created between them.

In short, the authentication server creates the Tunnel PAC and gives it to the peer. Then during authentication phase 1, the peer just sends back the PAC-Opaque portion of the Tunnel PAC to the authentication server.

In the same manner, User Authorization PAC is also created by the authentication server and it is also opaque to the peer which means the peer does not understand what is in it and it cannot interpret it. The peer just sends it to the server within phase 2, to authenticate itself to the server. In this case, any inner authentication method

may be skipped. It should be noted that User Authorization PAC does not include PAC-Key. Thus it should be bounded to the Tunnel PAC (Cam-Winget, McGrew, Salowey & Zhou, 2009). Figure 3.6 illustrates the usage of Tunnel PAC as well as User Authorization PAC.

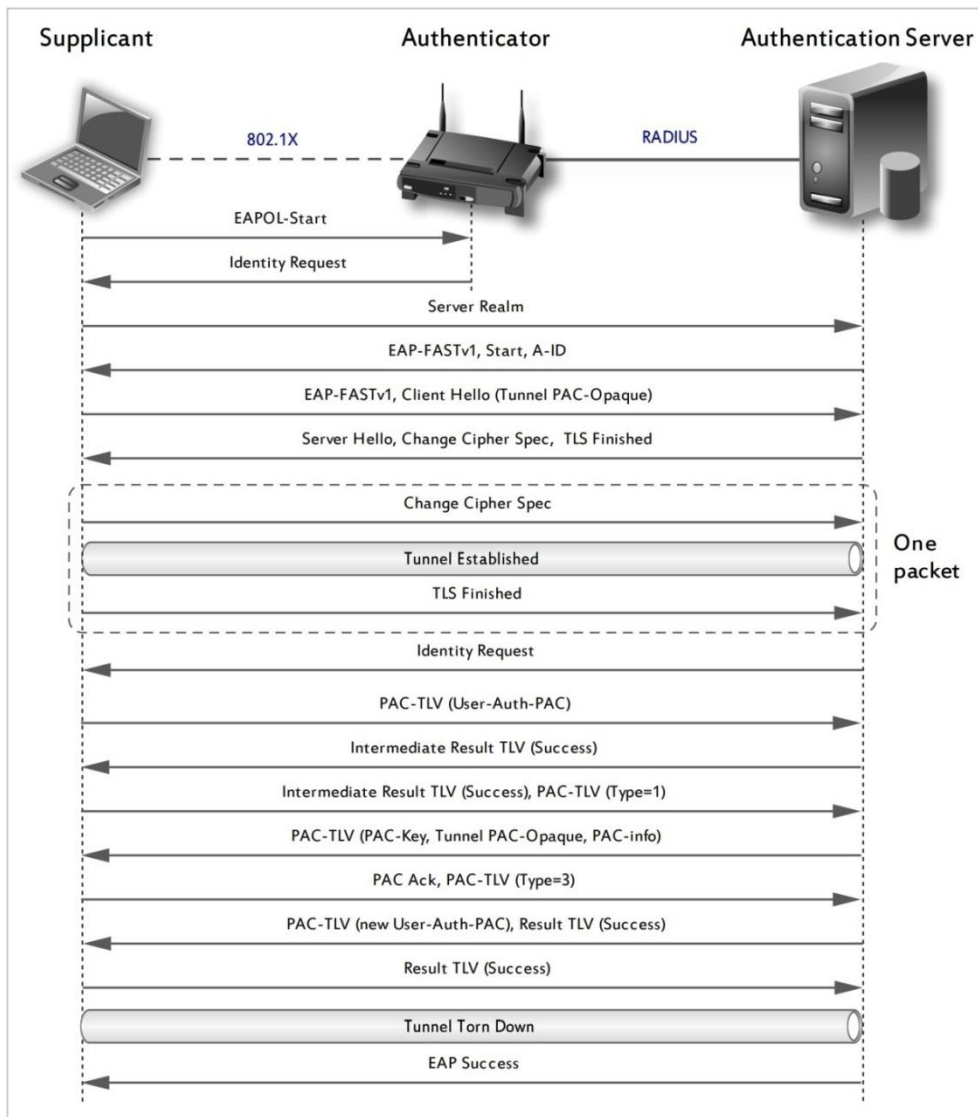


Figure 3.6 EAP-FAST authentication: User-Authentication-PAC usage.

3.5.3 EAP-FAST Provisioning Modes

- *Server-Authenticated Provisioning Mode:* The protected tunnel is established using server-side certificates (Figure 3.7).
- *Server-Unauthenticated Provisioning Mode:* The protected tunnel is established based on anonymous Diffie-Hellman key exchange (Figure 3.8).

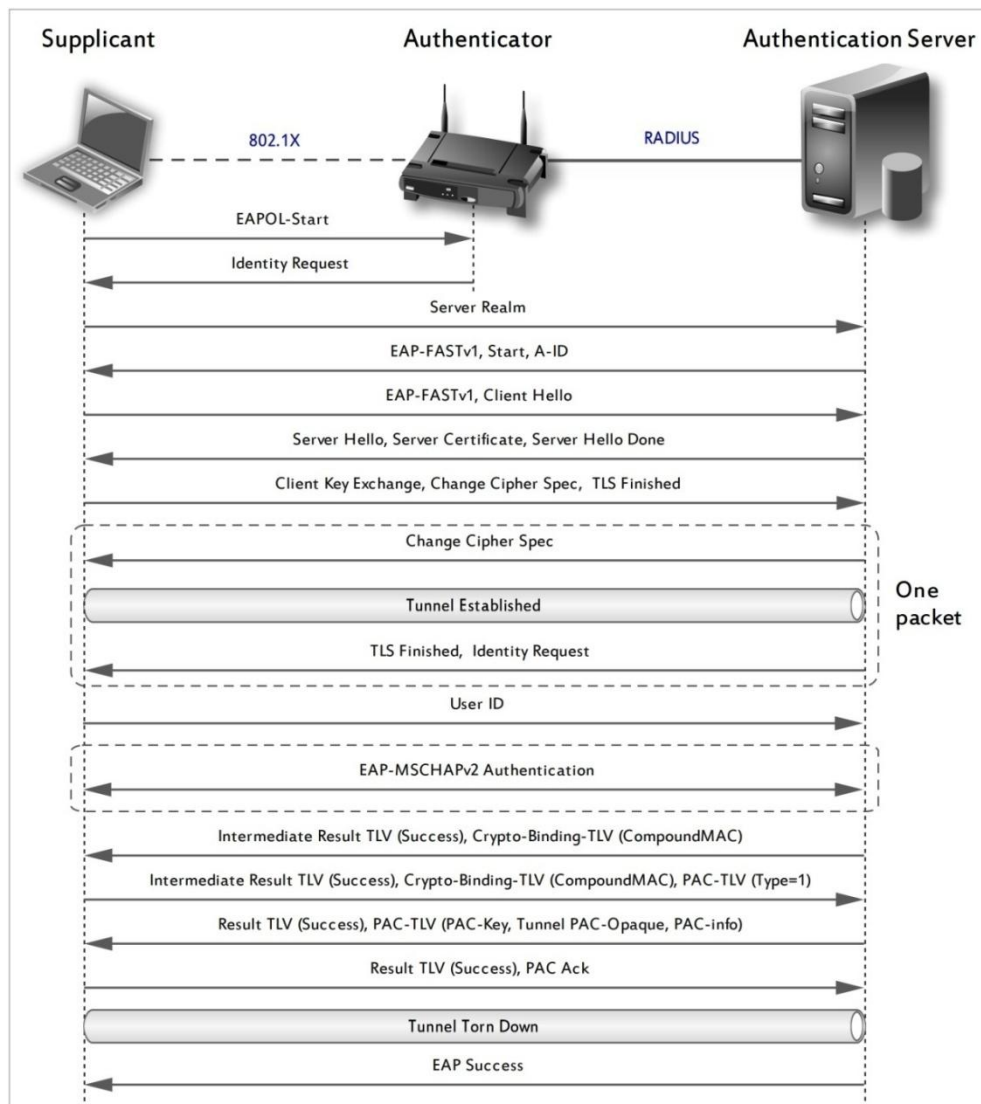


Figure 3.7 Server-authenticated dynamic provisioning.

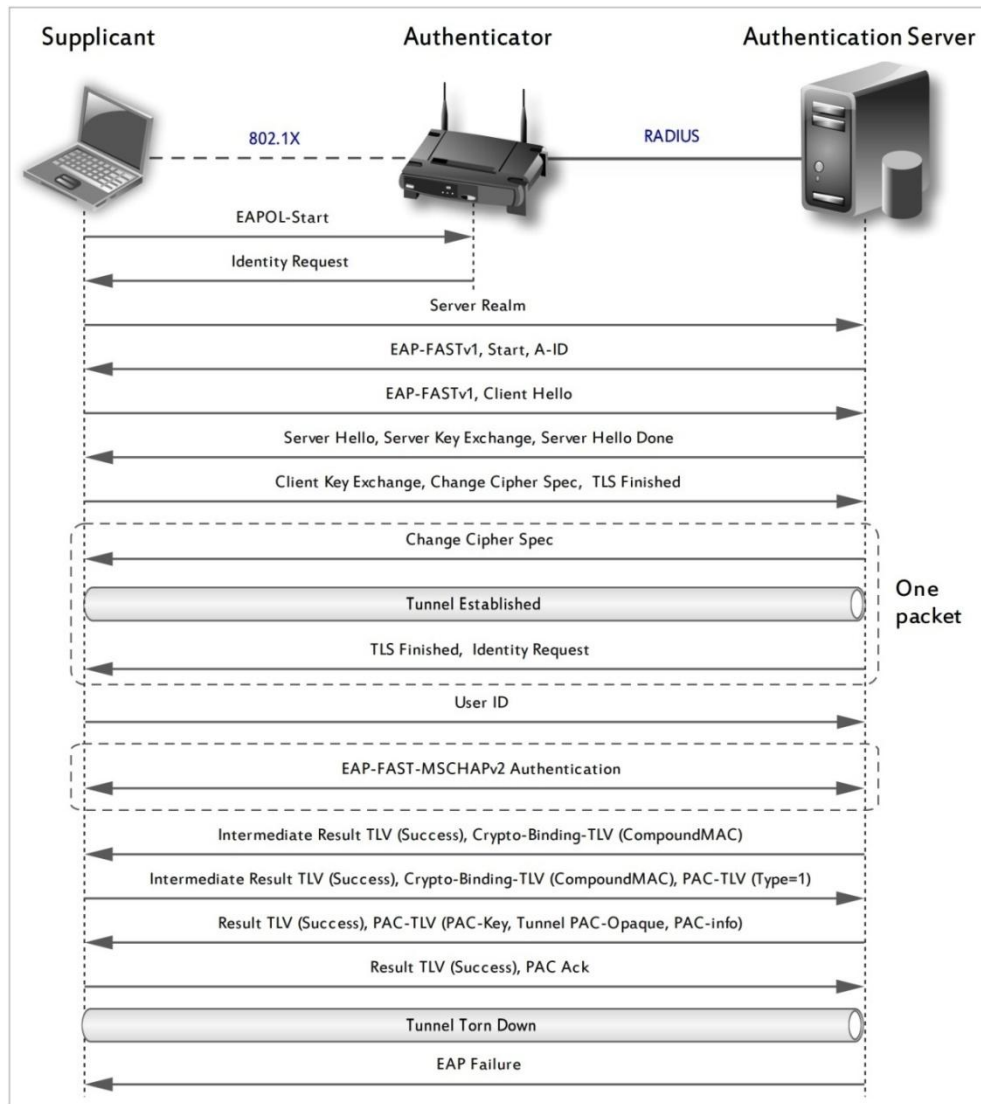


Figure 3.8 Server-unauthenticated dynamic provisioning (Cam-Winget, McGrew, Salowey & Zhou, 2009).

3.5.4 MITM on Tunnel-Based EAP Methods

Asokan, Niemi & Nyberg (2002) describe the vulnerability of tunnel-based EAP methods to man-in-the-middle attack. This attack can be launched as follows:

An adversary, acting as a peer, initiates a tunnel-based EAP method with the authentication server. The adversary executes a tunnel protocol with the authentication server in which the authentication server authenticates to the adversary. As a result of a successful tunnel protocol execution, both the adversary

and the authentication server obtain tunnel key (TK). The server then initiates an inner authentication method inside the protective tunnel. The adversary, acting as an authentication server, initiates a parallel session with a peer using the same authentication method outside a tunnel. The adversary then replays the peer's response into the tunnel, making the authentication server believe that the messages are coming from the other end of the tunnel. Thus, the inner authentication method, and the tunnel-based EAP method are executed successfully, and both the adversary and the authentication server subsequently share the established MSK if it is derived from the tunnel key (TK). Figure 3.9 shows the man-in-the-middle attack against tunnel-based EAP methods.

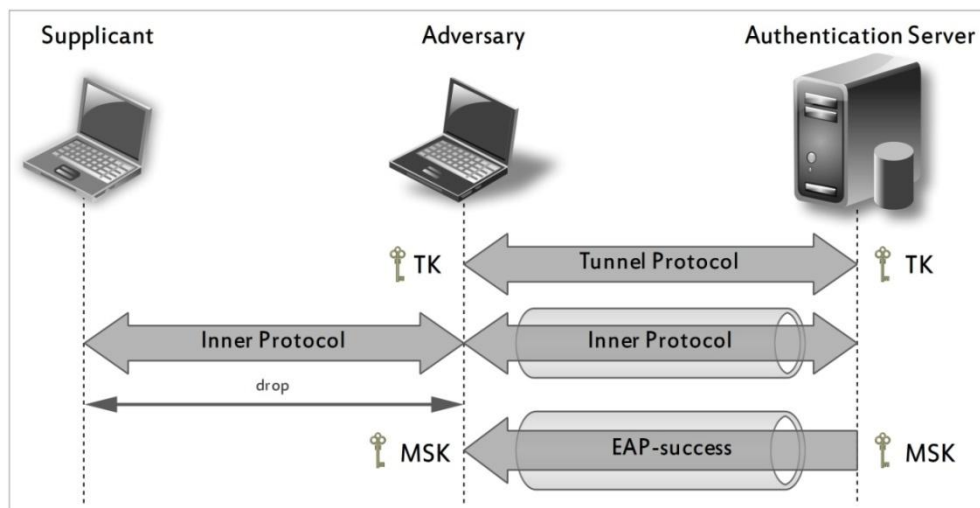


Figure 3.9 Man-in-the-middle attack on tunnel-based methods (Hoeper & Chen, 2009).

3.5.5 EAP-FAST MiTM Attack Protection

EAP-FAST provides protection from aforementioned man-in-the-middle attacks in two ways (Cam-Winget, McGrew, Salowey & Zhou, 2007):

1. *By using the PAC-Key:* In phase 1, the tunnel PAC is not only used for server authentication but also server can authenticate peer through information found in tunnel PAC. Thus, mutually authentication mitigates the man-in-the-middle attack described above.

2. *By using the Crypto-Binding TLVs:* In phase 2, Crypto-Binding TLVs are used to bind the outer authentication protocols with inner authentication protocols through derived keys from both authentication methods. Crypto-Binding assures that the outer authentication and inner authentication is occurred between the same peer and the server.

3.5.6 Summary of EAP-FAST Features

- It provides not only strong security but also convenience and efficiency by using PACs. Since it uses shared secrets that have strong entropy, it is much more faster than PEAP and EAP-TTLS.
- Enables the network access communication to be computationally lightweight. Uses PAC in lightweight devices.
- PACs are unique to each client identity. A different client cannot use the same PAC file or authentication will fail.
- Using PAC, allows faster TLS tunnel establishment.
- Supports crypto-binding, mixing the tunnel encryption key with the inner EAP method key to prevent MITM attack.
- Supports anonymous provisioning and manual provisioning of PAC, eliminate the need for PKI or use of server certificate.
- Supports EAP inner method chaining.
- Supports authorization PAC to allow fast session resumption without server state, allowing endpoints to roam the sessions across multiple AAA servers (Salowey, Zhou, Eronen & Tschofenig, 2008).

3.6 Comparison of TLS-Based EAP Methods

Table 3.3 and Table 3.4 show an in-depth comparison of the TLS-based EAP methods.

Table 3.3 EAP Security Claims (Cam-Winget, McGrew, Salowey & Zhou, 2007, Funk & Blake-Wilson, 2008, Palekar & others, 2004, Simon, Aboba & Hurst, 2008).

| EAP Security Claims | EAP-TLS (RFC 5216) | EAP-TTLSv0 (RFC 5281) | PEAPv2 (Draft, 2004) | EAP-FASTv1 (RFC 4851) |
|-------------------------------------|-------------------------------|----------------------------------|---------------------------------|----------------------------------|
| Ciphersuite negotiation | Yes | Yes | Yes | Yes |
| Mutual authentication | Yes | Yes | Yes | Yes |
| Integrity protection | Yes | Yes | Yes | Yes |
| Replay protection | Yes | Yes | Yes | Yes |
| Confidentiality | Yes | Yes | Yes | Yes |
| Key derivation | Yes | Yes | Yes | Yes |
| Key strength | Variable | Up to 384 bits | Variable | Variable |
| Dictionary attack protection | Yes | Yes | Yes | Yes |
| Fast reconnection | Yes | Yes | Yes | Yes |
| Cryptographic binding | N/A | No | Yes | Yes |
| Session independence | Yes | Yes | Yes | Yes |
| Fragmentation | Yes | Yes | Yes | Yes |
| Channel binding | No | No | No | No |

Table 3.4 Summary of TLS-based EAP methods (Coleman, Westcott, Harkins & Jackman, 2010).

| Features | EAP-TLS (RFC 5216) | EAP-TTLSv0 (RFC 5281) | PEAPv2 (Draft, 2004) | EAP-FASTv1 (RFC 4851) |
|---------------------------------|-------------------------------|----------------------------------|---------------------------------|----------------------------------|
| Server authentication | Certificate | Certificate | Certificate | PAC |
| Client authentication | Certificate | Any method | Any EAP method | Any EAP method |
| Server certificate | Required | Required | Required | Optional |
| Client certificate | Required | Optional | Optional | Optional |
| Tunnel establishment | Optional | Necessary | Necessary | Necessary |
| User identity protection | No | Yes | Yes | Yes |
| Ease of deployment | Hard | Moderate | Moderate | Moderate |
| Security strength | Highest | Medium | High | High |

CHAPTER FOUR THE AVISPA TOOL

To validate the EAP-FAST protocol we used the automatic protocol analyzer AVISPA (Armando & others, 2005). Its good expressive form and ease-of-use are the attractive features of the tool, but the main advantage of AVISPA is the ability to use different verification techniques on the same protocol specification. In AVISPA, security protocols are specified by High Level Protocol Specification Language (HLPSL). As indicated in Chevalier & others (2004), the HLPSL language has already proven itself to be an effective language for modeling security protocols: many protocols of varying levels of complexity. We have chosen AVISPA mainly because it is concluded as more efficient tool to falsify and verify security protocols than the other several widely used tools (Patel & others, 2010). Figure 4.1 depicts the classification of formal methods for security protocol analysis (Modersheim, Vigano & von Oheimb, 2005).

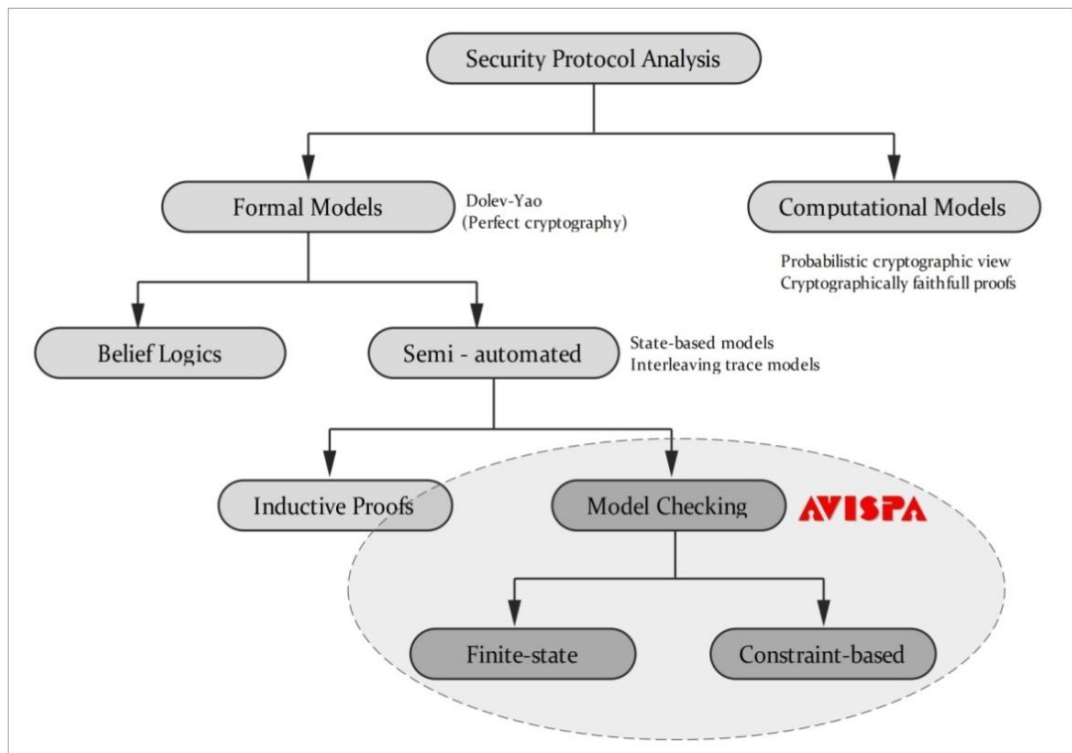


Figure 4.1 Protocol analysis techniques (Modersheim, Vigano & von Oheimb, 2005).

AVISPA (Automated Validation of Internet Security Protocols and Applications) is a research tool that automatically validates and analyzes formal models of security-sensitive protocols. In AVISPA (Automated Validation of Internet Security Protocols and Applications [AVISPA], 2006b), protocols and their security requirements are described using HLPSL language. A `hlpsl2if` translator takes as input a HLPSL specification and translates it into a corresponding Intermediate Format (IF) specification automatically. IF (AVISPA, 2003b) is a lower-level language than HLPSL and is read directly by the state-of-the-art back-ends embedded in AVISPA. The IF specification of a protocol is then analyzed by back-end tools to test if the security goals are satisfied or violated (Figure 4.2). If any attack is found back-ends return it in an intuitive and readable output format. The command-line AVISPA Tool outputs attack traces in an Alice&Bob notation. The web interface displays an attack trace in the form of a Message Sequence Chart (Figure 4.3).

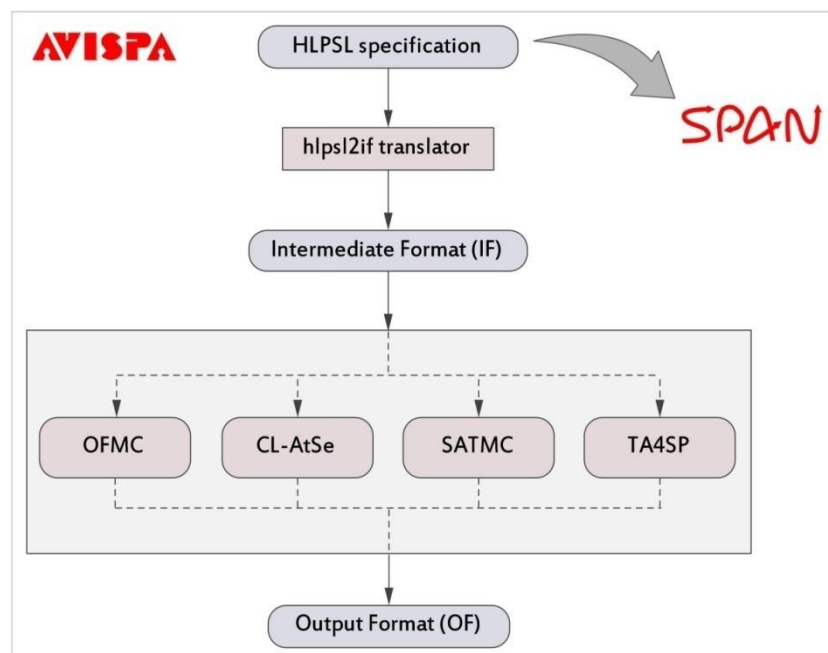


Figure 4.2 Architecture of the AVISPA tool (AVISPA, 2006b).

4.1 The High Level Protocol Specification Language (HLPSL)

HLPSL (AVISPA, 2003a) is a role-based language. It is easier to specify a protocol from Alice&Bob notation. Alice-Bob notation describes the security protocols using flow of messages between the involved parties (Figure 4.4).

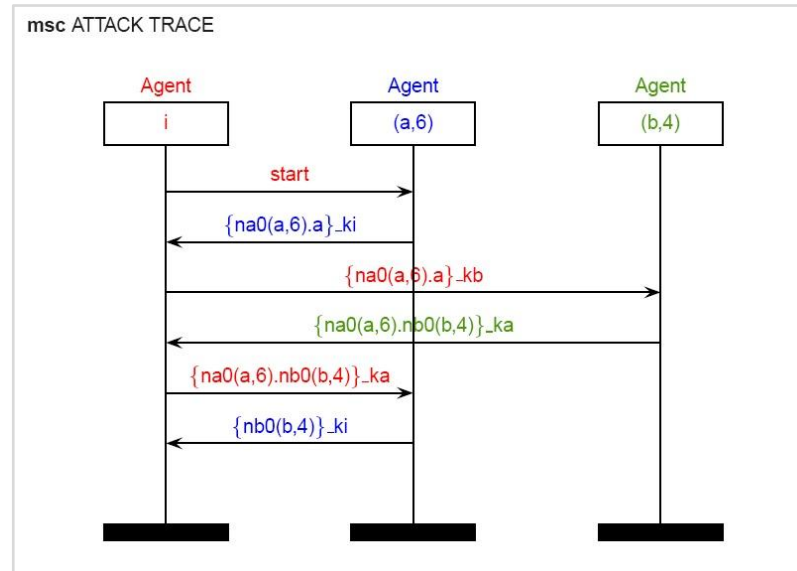


Figure 4.3 Attack trace of AVISPA web tool (Armando & others, 2005).

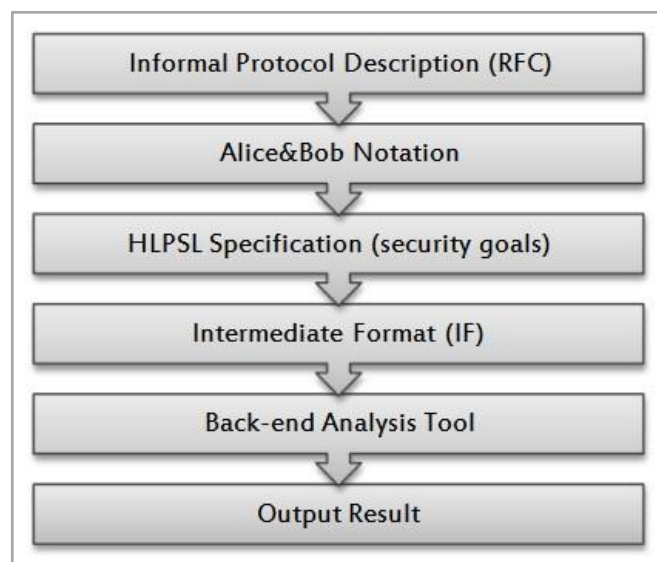


Figure 4.4 Analysis steps using AVISPA.

The HLPSL consists of following sections:

- *Basic roles* specifies the initial knowledge and the behaviour of each honest participant in a protocol. Basic roles contain a set of transitions. Generally, each transition represents the receipt of a message and the sending of a reply message. A transition consists of a trigger, or precondition, and an action to be performed when the trigger event occurs.

- *Composed roles* instantiate the participants(basic roles) and specify how they interact with one another. Usually, roles are executed simultaneously (parallel) as independent state-machines.
- *Environment role* is a top level role which contains a composition of one or more sessions, where the intruder may play some roles as a legitimate user. It also contains global constants and describes what knowledge the intruder initially has.
- *Goals*: AVISPA supports different forms of authentication and secrecy. In this section, goals are specified by using predefined macros which are:
 - the secrecy of some information,
 - the strong authentication of agents on some information,
 - the weak authentication of agents on some information.

Each goal is identified by a constant, referring to predefined predicates (*secret*, *witness*, *request* and *wrequest*) declared explicitly in transitions. Here, '*witness*' and '*request*' pair specifies strong authentication while '*witness*' and '*wrequest*' pair used for weak authentication. One pair of witness/request serves for unilateral authentication, so for mutual authentication there should be defined two pairs. When a sender sends any message, he/she may issue a '*witness*' to denote that he/she needs the message to be correctly delivered to the receiver. On the other hand, the receiver should issue a '*request*' to answer the '*witness*' (AVISPA, 2006a). For instance,

- *witness(A,B,authNonce,Nonce')*
means "'A' wishes to prove his identity to 'B', and presents Nonce'. 'A' wants to ensure that 'B' received the exact value of Nonce' that is sent".
 - *request(B,A,authNonce,Nonce')*
means " 'B' authenticates 'A' on Nonce', i.e. 'B' wants to ensure that 'A' sent this value of the Nonce' which is received".
- Here; 'A' and 'B' are agents, 'authNonce' is a constant 'protocol_id' identifying the 'witness-and-request' statements in the goals section.

The main rule to remember when putting 'requests' is to put them as late as possible. On the other hand since the secrecy check takes effect only after the events have been issued, the 'secrecy' events should be given as early as possible i.e. right when the secret term has been created in the respective role transition.

The above described 'witness-and-request' predicates are defined in transitions of basic roles. In the 'Goal' section those properties specified as following:

```
goal
  secrecy_of na          % na is a constant 'protocol_id' representing secret term
  authentication_on authNonce
end goal
```

AVISPA mainly covers the following Goals, but also several other goals may be approximated (AVISPA, 2003c):

- *Authentication* (unicast and multicast)
 - Entity authentication
 - Message origin and integrity
 - Replay protection
- *Key agreement* (reduced to authentication)
 - Key authentication
 - Key confirmation
 - Fresh key derivation
- *Confidentiality* (Secrecy)

Basic types in HLPSL:

- *agent* : names of principles
- *public_key* : asymmetric keys
- *symmetric_key* : symmetric keys
- *nat* : natural numbers

- *hash_func* : to model hash functions etc
- *bool* : boolean values for modeling flags
- *channel(dy)* : for exchanging messages. The intruder is modeled by the 'dy' channel (the Dolev-Yao intruder) over which the communication takes places. Communication in HLPSL is synchronous, via immediate transitions.

HLPSL supports cryptographic primitives such as nonces, hash functions, signatures, encryption, etc. and algebraic properties like concatenation ('.'), exclusive or (*xor()*), exponential (*exp()*) (AVISPA, 2006a).

4.2 The Dolev-Yao Intruder

AVISPA implements the Dolev-Yao intruder model. Dolev-Yao is known as the most general and the strongest possible intruder model for formal protocol analysis (Cervesato, n.d.). Under this model, the intruder has full control over the network, meaning that each message received by a participant has also been received by the intruder. The model (Dolev & Yao, 1983) states that the active intruder has the capability to :

- read all messages
- block any message
- arbitrarily re-direct messages
- store messages it receives indefinitely
- build new messages with the different constructors
- arbitrarily re-order messages
- decompose messages into their components
- encrypt/decrypt messages and modify them if it possesses the appropriate key (Black-box perfect crypto)

The only restriction that is placed on the Dolev-Yao intruder is that it cannot break encryption. If it receives an encrypted message, it cannot learn the contents of the message unless it has knowledge of the appropriate key.

Naturally, it is hardly realistic that a process performing all of a Dolev-Yao attacker's actions is active on a physical network, but if a protocol can withstand a Dolev-Yao attacker, then it is reasonably certain that any real-world attacks will fail as well.

4.3 The Back-End Analyzers

OFMC (On-the-fly Model-Checker) is based on two lazy techniques: the first is lazy demand-driven search and second is the lazy intruder, which reduces the computational effort. Lazy demand-driven search uses lazy data types to model infinite state-space of protocol. Lazy data types model the protocol and attacker as infinite tree on the fly, in a demand driven way. The nodes of the tree are traces and children represent the next step of protocol or an action of an attacker. Properties of nodes represent the security properties. Lazy intruder techniques model a lazy Dolev-Yao intruder whose actions are generated in a demand-driven way. Now, the OFMC is renamed as Open source Fixed-point Model-Checker (Basin, Modersheim & Vigano, 2005).

CL-AtSe (Constraint-Logic-based Attack Searcher) is OCaml-based (programming language) implementation of the deduction rules. These rules allow user to interpret and automatically execute the protocols in every possible way in the presence of Dolev-Yao intruder Capabilities. The main design goals of CL-Atse are modularity (easily extend the class of the protocols to be analyzed) and performance (obtain the results using large number of protocol sessions). The analysis algorithm used by CL-AtSe is designed for a bounded number of loops, i.e. a bounded number of protocol steps in any trace. Any state-based properties (like secrecy, authentication etc) and algebraic properties of operators like XOR, exponentiation can be modeled and analyzed (Turuani, 2006).

SATMC (SAT-based Model-Checker) is an open and flexible platform for SATbased bounded model checking. Protocol descriptions are specified as rewrite formalism in IF format. SAT compiler generates the formula for each step of the

protocol using encoding techniques. Each formula is then tested using SAT solver - whether formula is satisfiable or it leads to an attack. SATMC performs bounded analysis by considering finite sessions of the protocol with Dolev-Yao intruder capabilities (Armando & Compagna, 2004).

TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols): is based on abstraction-based approximation method. Abstraction provides a way to prove correctness or security of a protocol by over-estimating the possibility of failure. This tool language represents an over-approximation or under-approximation of the intruder knowledge with an unbounded number of sessions. For secrecy properties, TA4SP can show whether a protocol is flawed (by under-approximation) or whether it is safe for any number of sessions (by over-approximation) (Boichut, Heam, Kouchnarenko & Oehl, 2004).

4.4 The SPAN Tool

SPAN (a Security Protocol ANimator for AVISPA) is an animation tool that makes HLPSL specification debugging more easy. SPAN allows us to have a better understanding of the specification, check that it is executable and that it corresponds to what is expected (Figure 4.5). From an HLPSL specification, it is possible to interactively produce a Message Sequence Chart (MSC) corresponding to an execution of the specification step by step (Figure 4.6) (Glouche, Genet & Houssay, 2008). AVISPA is very convenient, especially when visualized with SPAN. Here are some features of SPAN:

- Supports the editing of protocol specifications.
- Allows to select and configure the back-ends integrated into the tool.
- It is possible to go back in the execution.
- Hide/show content of variables of roles.
- Can represent one or more sessions of the protocol in parallel.

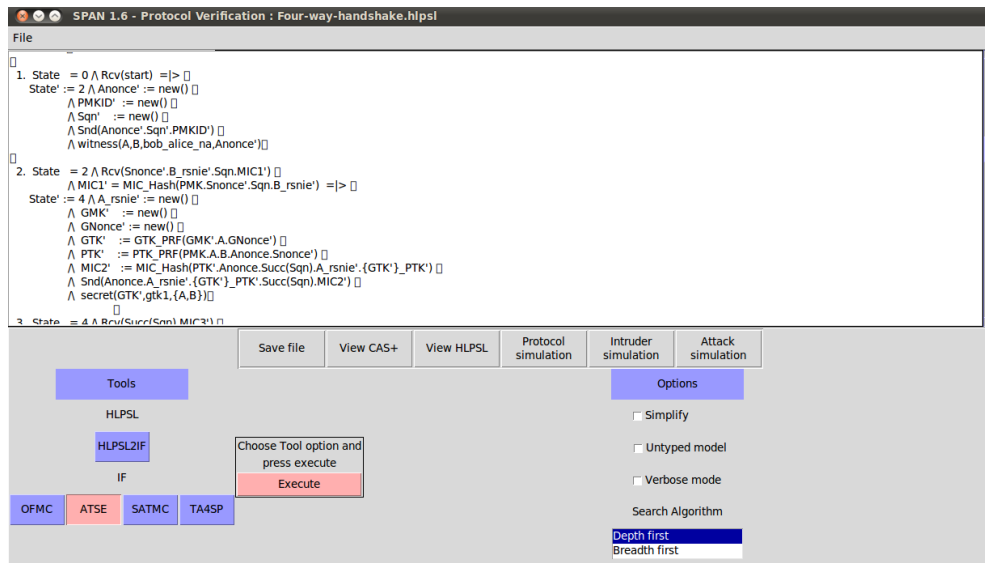


Figure 4.5 SPAN animator for AVISPA.

SPAN allows to launch three different modes:

- *Protocol Simulation* for simulating the protocol and build a particular MSC corresponding to the HLPSSL specification (Figure 4.6).
- *Intruder Simulation* for simulating the protocol with an active/passive intruder (Figure 4.7).
- *Attack Simulation* for automatic building of MSC attacks from the output of either OFMC or CL-ATSE tools.

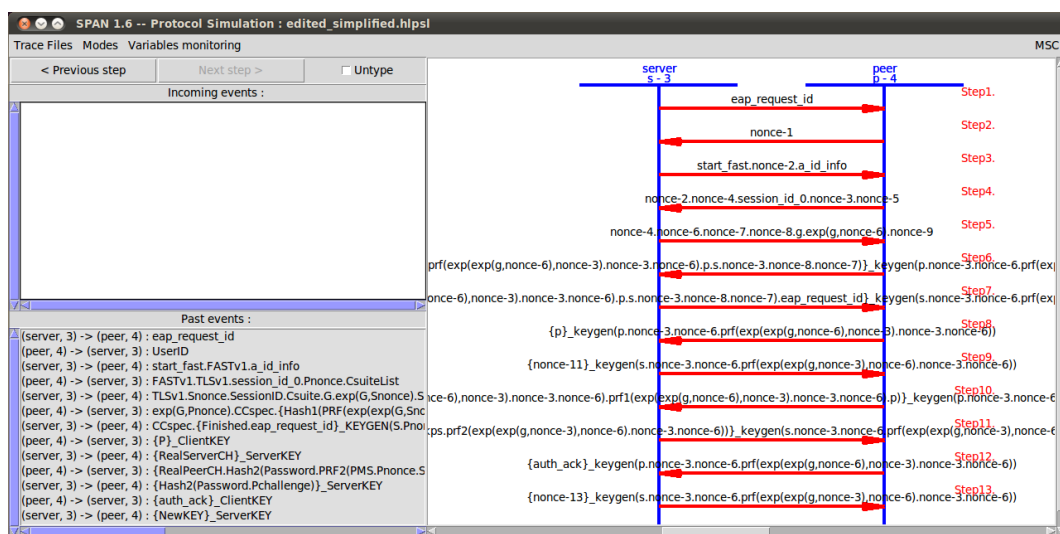


Figure 4.6 Protocol simulation in SPAN.

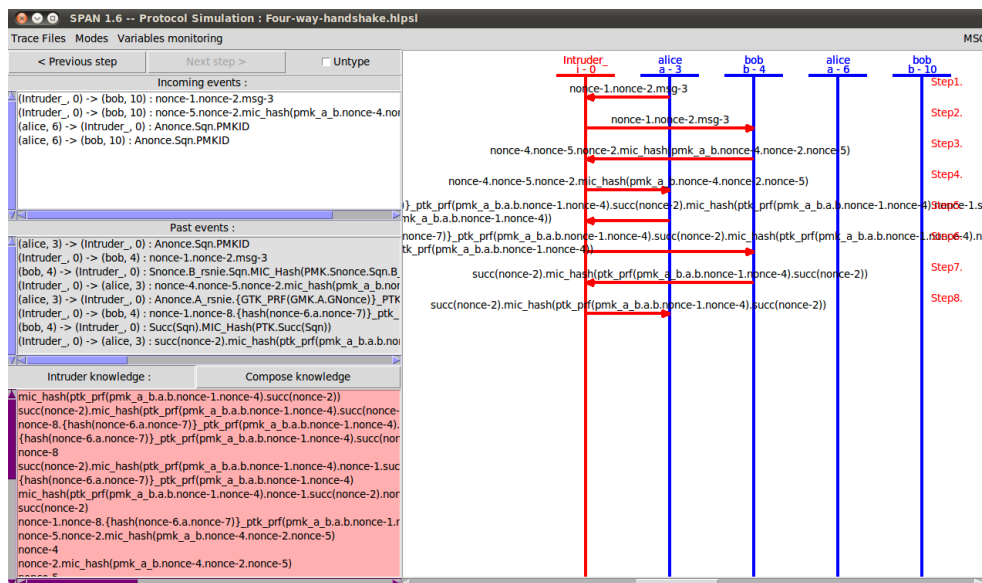


Figure 4.7 Intruder simulation in SPAN.

4.5 Summary of AVISPA Features

- *Usefulness:* The main advantages of AVISPA are its good expressive form and ease of use.
- *Scalability and flexibility:* AVISPA has been successfully validated a number of security protocols developed by the IEEE, IETF Working Groups, and others. The AVISPA Library is the best publicly available library that comprises a large collection of security protocols, specified together with their properties in HLPSL (AVISPA, 2005)
- *Availability:* Freely available web interface to experiment the AVISPA Tool (Figure 4.8). It can be accessed at the URL, <http://www.avispa-project.org/web-interface>.
- *Visualization:*
 - Provided a very helpful HLPSL XEmacs mode. Its syntax highlighting and menus are very practical for editing protocol specifications (Figure 4.9).
 - A Security Protocol ANimator for AVISPA (SPAN) helps in interactively producing Message Sequence Charts from an HLPSL specification.
- *High performance:* AVISPA is capable of analyzing the model and its properties in a short period of time.



Figure 4.8 AVISPA's web interface.

```

emacs: H.530.hlpsl
File Edit View Cmds Tools Options Buffers AVISPA Help
role VisitedGateKeeper (
  MT,VGK,AuF : agent,
  SND,RCV   : channel(dy),
  F         : function,
  ZZ_VA    : symmetric_key,
  NIL,G     : text)
played_by VGK def=
  local
    State      : nat,
    GK,Key,Key1,FM1,FM2,FM3,M2 : message,
    Y,CH2,CH4  : text (fresh),
    CH1,CH3    : text
  init State = 0
  transition
  1. State = 0 /\ RCV(MT.VGK.NIL.CH1'.GX'.FM1') =|>
     State' = 1 /\ Key' = exp(GX',Y')
                /\ M2' = MT.VGK.NIL.CH1'.GX'.FM1'.VGK.xor(GX',exp(G,Y'))
                /\ SND(M2'.F(ZZ_VA.M2'))
                /\ witness(VGK,MT,key,Key')
  2. State = 1 /\ RCV(VGK.MT.FM2'.FM3'.F(ZZ_VA.VGK.MT.FM2'.FM3')) =|>
     State' = 2 /\ SND(VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'.
                    F(Key.VGK.MT.CH1.CH2'.exp(G,Y).FM3'.FM2'))
  3. State = 2 /\ RCV(MT.VGK.CH2.CH3'.F(Key.MT.VGK.CH2.CH3')) =|>
     State' = 3 /\ SND(VGK.MT.CH3'.CH4'.F(Key.VGK.MT.CH3'.CH4'))
                /\ request(VGK,MT,key1,Key)
                /\ secret(Key,MT)
end role
IS08-----XEmacs: H.530.hlpsl (HLPSL -- AVISPA Font)----L1--G0--All--

```

Figure 4.9 XEmacs mode of AVISPA (Armando & others, 2005).

4.6 AVISPA Modeling Limitations

The HLPSL language is simple and capable of expressing most authentication and key exchange protocols, but also have a number of limitations:

- Lack of support algebraic equations (arithmetic expressions such as '+', '-', '*', '/', '>', '<', "mod")
- Lack of support for fairness constraints, timestamps, timeouts and delays.
- Could not be used to model security protocols with non-repudiation requirements.
- Does not support anonymity goals (like; identity privacy, location privacy)
- Supports only a single intruder model; the Dolev-Yao intruder model. This is the most powerful intruder model, and is generally the type of intruder which protocols are designed to be secure against. However communication mediums are becoming more diverse and this model is no longer suitable in all cases.
- Cannot detect attacks such as DoS attacks, Guessing attacks, Downgrade attacks, Dictionary attacks, Brute-force attacks.

4.7 AVISPA Usage Recommendations

- Check the executability of the protocol before run it against the security properties to find attacks. Use the step-by-step simulation of the protocol to see that the specification is what it should be and all the states are reachable. Examples of checking the executability (runnable) of HLPSSL specification is given below:

```
avispa protocol_model.hlpsl --cl-atse -noexec
```

```
avispa protocol_model.hlpsl --ofmc -sessco
```

```
avispa protocol_model.hlpsl --satmc --check_only_executability=true
```

- Tools stop the search at the first attack if they find out. So it is recommended to drop the goal that was found violated, to see if the other goals of the protocol do hold.
- It is not possible to disable the intruder completely, because AVISPA relies on him to relay messages. It is very useful to specify only one session, between only honest agents, and then run to see how protocol works.
- It is preferred to use compound types rather than the most general type 'message'. For instance, if $X = \{Na.Nb\}_{Kab}$; where Na, Nb are nonces and Kab is a symmetric key, X should be declared as *{text.text}_symmetric_key*.

CHAPTER FIVE

VALIDATION OF PROTOCOLS

We have modeled the Dynamic Provisioning mechanisms, EAP-FAST authentication mechanisms and the Four-Way Handshake protocol in HLPSL. In this chapter, we will show the A&B notations of each protocol and output results of AVISPA tool. A&B notation shows a clear illustration of the messages exchanged in a normal run of a given protocol (AVISPA, 2006a). It is convenient and very helpful to get protocols in the form of flow of messages before specify them in HLPSL.

AVISPA has its web interface just as a demo for presenting the tool. Because, the resources (memory and time) assigned to each test are significantly limited on it. We used local version of AVISPA: AVISPA-1.1 and SPAN-1.6 on UBUNTU 10.04.

The statistics given by each tool are not uniform. There are some timings, for parsing/translating the HLPSL specification into a back-end usable specification, and for the total execution time. Each backend has its own output format. For instance OFMC output results mean:

| | |
|--------------------------------|---|
| parseTime: 0.00s | > the time for reading the input file |
| searchTime: 0.27s | > the time for the analysis of the protocol |
| visitedNodes: 119 nodes | > the number of nodes in the tree |
| depth: 8 plies | > the depth of the tree |

In the analysis of protocols, we mainly used Cl-Atse model checker supported by AVISPA. Because in Vigano (2006), the CL-AtSe has shown better properties than other model checkers, implemented in the tool, such as OFMC, SATMC or TA4SP.

EAP-FAST authentication process occurs in three phases. Phase 0 is an optional phase in which the PAC can be provisioned manually or dynamically. PAC provisioning is only done once to set up the PAC secret between the server and client and all subsequent EAP-FAST sessions skip "Phase 0". In Phase 1, the peer and the

authentication server uses the PAC to establish TLS tunnel. In Phase 2, the peer authenticates to the server by another EAP method inside the encrypted tunnel. Figure 5.1 depicts the EAP-FAST process.

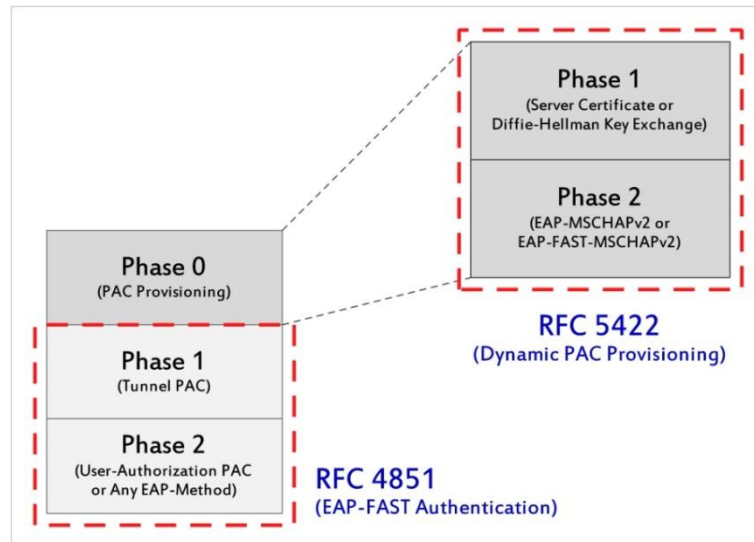


Figure 5.1 EAP-FAST authentication phases.

5.1 Dynamic Provisioning using EAP-FAST

Dynamic Provisioning occurs in the Phase 0 of EAP-FAST protocol. The Phase 0 is independent of other phases which may be skipped in the case of the peer has appropriate PACs. There are two modes of Dynamic PAC provisioning:

- Server-Authenticated Provisioning
- Server-Unauthenticated Provisioning

In both modes, only Tunnel PAC is allowed to be provisioned (Cam-Winget, McGrew, Salowey & Zhou, 2009).

5.1.1 Server-Authenticated Provisioning Mode

In this mode, the secure tunnel is established using the TLS handshake protocol and within the tunnel the peer is authenticated to the server with EAP-MSCHAPv2 protocol. The server is authenticated to the peer twice: in phase 1 by certificates and

in phase 2 by challenge/responses. A Figure 5.2 illustrates the A&B notation of the Server-Authenticated provisioning mode of EAP-FAST protocol (Cam-Winget, McGrew, Salowey & Zhou, 2009).

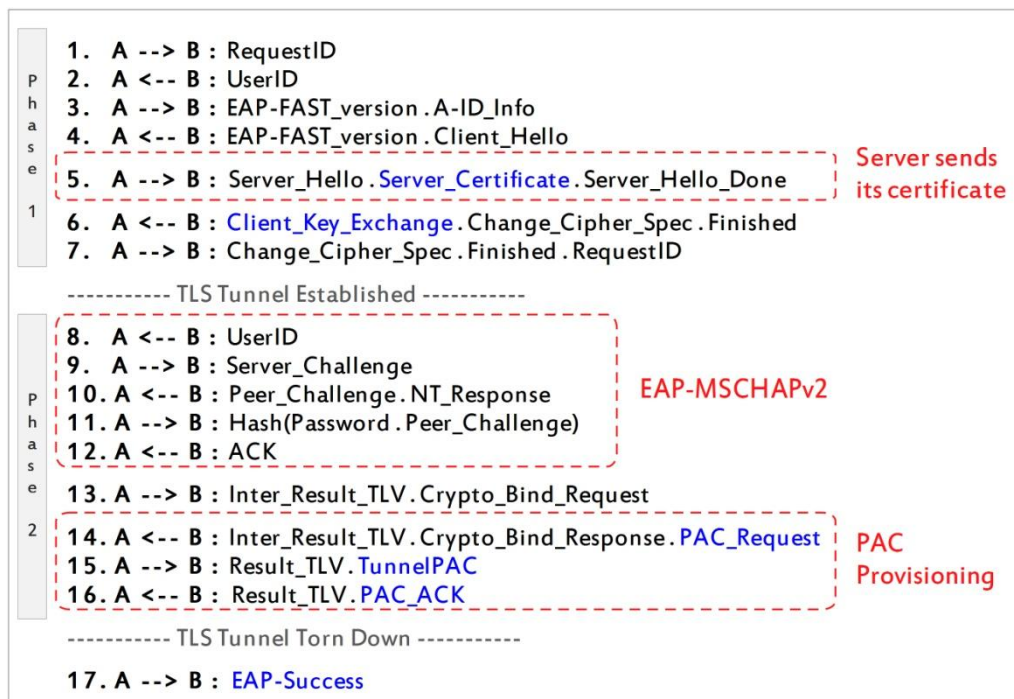


Figure 5.2 Server-authenticated provisioning.

where,

- A and B : Server and Peer respectively
- A-ID_Info : Server realm or hint
- Kserver : Server's Public key
- Kca : Certificate Authority's Public key
- Client_Hello : TLS_version.Session_ID.Peer_nonce.CiphersuiteList
- Server_Hello : TLS_version.Session_ID.Server_nonce.Ciphersuite
- Server_Certificate : $\{A.K_{server}\}_{inv}(K_{ca})$
- Server_Hello_Done : Informative message
- Client_Key_Exchange : $\{PMS\}_{K_{server}}$
- PMS : Pre-Master-Secret = Randomly generated value by Peer
- Change_Cipher_Spec : Informative message
- Finished : Encrypted hash of all previous messages with MS

MS : Master-Secret = PRF(PMS.Peer_nonce.Server_nonce)
 Client_Key : Client Session Key = PRF0(B.Peer_nonce.Server_nonce.MS)
 Server_Key : Server Session Key = PRF0(A.Peer_nonce.Server_nonce.MS)

%% Within tunnel all messages are encrypted with Server's or Peer's session keys

Server_Challenge : Randomly generated value by Server
 Peer_Challenge : Randomly generated value by Peer
 Password : Shared Secret between the Server and the Peer
 NT_Response : Hash(Password.(Peer_Challenge.Server_Challenge.User_ID))
 Inter_Result_TLV : Success or Failure
 Result_TLV : Success or Failure
 PAC_Request : PAC type
 TunnelPAC : PAC_key.PAC_opaque.PAC_info
 PAC_key : Shared Secret between Server and Peer
 PAC_info : Necessary information about PAC
 SMK : Server's Master Key
 PAC-opaque : {PAC_Key.PAC_info}_SMK
 Crypto_Bind_Request :
 Bind_version.FAST_version.zero.CMK_Nonce.Compound_MAC1
 Crypto_Bind_Response :
 Bind_version.FAST_version.one.CMK_Nonce.Compound_MAC2

Generating inner method (MSCHAPv2) MSK (Microsoft Corporation, 2001, Zorn, 2000, 2001):

MasterKey = HMAC(Hash(Hash(Password)), NT_Response)
 MasterSendKey = PRF1(MasterKey)
 MasterReceiveKey = PRF2(MasterKey)
 MSK = MasterReceiveKey.MasterSendKey

%Calculating Compound_MAC (Cam-Winget, McGrew, Salowey & Zhou, 2007):

Seed_Label : "Seed"

CMK_Label : "CMK"

Seed : Session Key Seed = PRF0(MS, Seed_Label, Peer_nonce.Server_nonce)

CMK : Compound Session Key = PRF(Seed, CMK_Label, MSK)

Compound_MAC1 =

HMAC(CMK, Bind_version.FAST_version.zero.CMK_Nonce)

Compound_MAC2 = HMAC(CMK, Bind_version.FAST_version.one.CMK_Nonce)

In Step2, the UserID is a bogus user ID. It may be "realm" or "anonymous". But in step 8, it is the real user ID.

In the end of the protocol run, in step 17, the server allowed the peer to access the network. Here, it should be noted that the server may also deny the access even if the authentication and provisioning were successful. It is up to network policy of the authentication server.

The Server-Authenticated provisioning is validated against the goals shown in Figure 5.3 and the output of the validation is shown in Figure 5.4. It is important to note that, only EAP methods that provide mutual authentication and key derivation should be used within the tunnel. Otherwise, this mode will be vulnerable to man-in-the-middle attack introduced in Asokan, Niemi & Nyberg (2002).

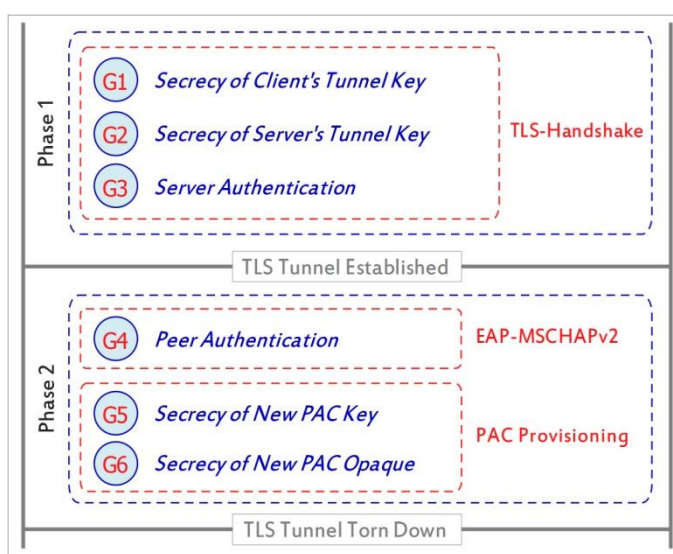


Figure 5.3 Validation goals of server-authenticated mode.

```

SUMMARY
SAFE

BACKEND
CL-AtSe

STATISTICS

Analysed   : 632832 states
Reachable  : 265869 states
Translation: 0.72 seconds
Computation: 85.82 seconds

```

Figure 5.4 The output results.

To validate the protocols we used CL-AtSe analyzer which does falsification. The CL-AtSe analyzer searches for attacks that falsify the goals of the protocol. There are two possible results that the analyzer may bring us: SAFE or UNSAFE. A protocol validated with AVISPA is SAFE or UNSAFE within the scope of security goals and the analysis scenario given.

5.1.2 Server-Unauthenticated Provisioning Mode

This mode enables any wireless client to be provisioned with new PAC. The server-side certificate is not necessary to establish the encrypted tunnel. The tunnel is established through anonymous handshake by Diffie-Hellman key exchange protocol. The clients, that don't have valid information to authenticate the server, can use this alternative provisioning mode to get PACs. After getting provisioned with PACs, the wireless clients have to reauthenticate to the network with new PACs.

The differences between this mode and the Server-Authenticated mode are the followings:

- The secure tunnel is established using the Diffie-Hellman Key Exchange protocol. Instead of using server-side certificates, the both parties exchange some seed materials to generate master keys separately:
 - G : Public value
 - $Server_Key_Exchange$: $G.exp(G,Server_nonce)$
 - $Client_Key_Exchange$: $exp(G,Peer_nonce)$
 - $PMS = exp(exp(G,Server_nonce),Peer_nonce)$

- The inner authentication protocol is EAP-FAST-MSCHAPv2 protocol, in which the server and the peer challenges are not transferred to each other. Both parties generate the challenges on themselves. Only the hash of challenges are exchanged.
 - Server_challenge : PRF1(PMS.Server_nonce.Peer_nonce)
 - Peer_challenge : PRF2(PMS.Server_nonce.Peer_nonce)
- In the end of the protocol run, in step 17, the server denies the access to the network, even if the authentication and provisioning were successful.

The Server-Unauthenticated provisioning mode is validated against the same goals as Server-Authenticated provisioning mode. A Figure 5.5 depicts the A&B notation of the Server-Unauthenticated provisioning mode (Cam-Winget, McGrew, Salowey & Zhou, 2009).

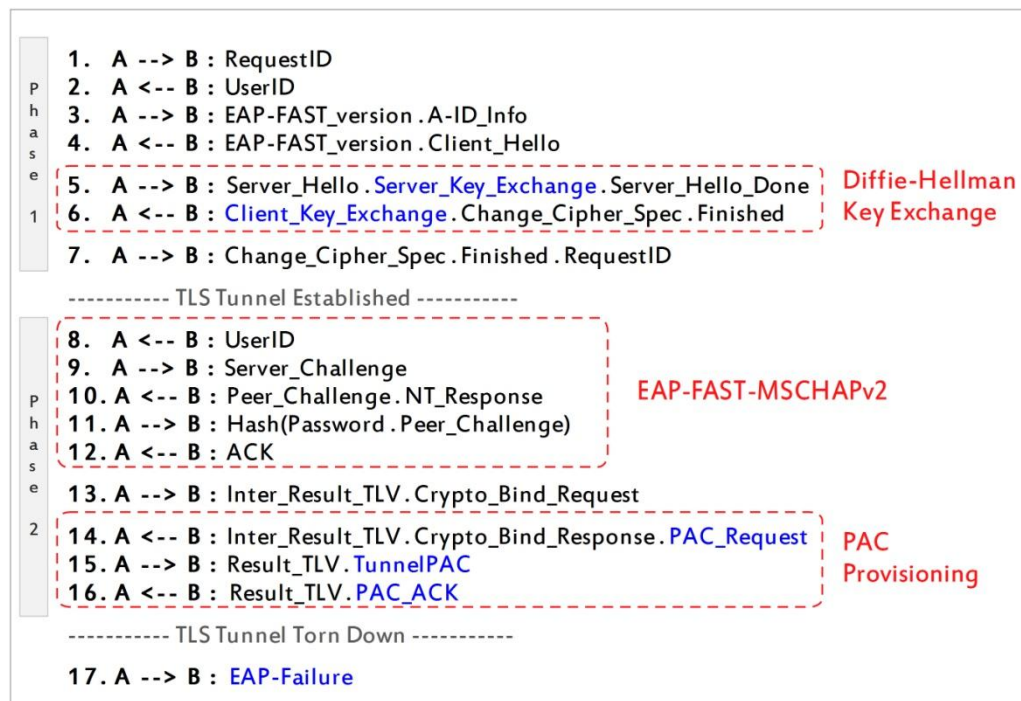


Figure 5.5 Server-unauthenticated provisioning.

Since the Server-Unauthenticated provisioning mode is using anonymous Diffie-Hellman key exchange protocol which is vulnerable to MiTM attack, AVISPA also

found this attack (Figure 5.6). Cam-Winget & others (2009) states that the Server-Unauthenticated provisioning mode is able to detect the MiTM attacks by two techniques:

- Using EAP-FAST-MSCHAPv2 method. Unlike EAP-MSCHAPv2, in this method, the peer and the server challenges are derived separately in both parties as part of the tunnel key derivation and they are not transferred to each other. It makes hard to launch active attacks. For this reason, only EAP-FAST-MSCHAPv2 can be used as inner authentication method in this provisioning mode.
- Cryptographically binding the keys derived from phase 1 with keys derived from phase 2.

However, in AVISPA, MiTM attack is not detected by aforementioned techniques (Figure 5.7). As expected, here MiTM is not generating EAP-FAST-MSCHAPv2 challenges but resending them. Also, MiTM just replays crypto-binding messages, as it cannot change the concept. As a result, MiTM was successful to disclosure of the PAC-key that is transmitted within the tunnel.

```

SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  TYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Server-unauth-prov.if

GOAL
  Secrecy attack on (n8(PAC_key))

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 108 states
  Reachable  : 41 states
  Translation: 0.77 seconds
  Computation: 0.00 seconds

```

Figure 5.6 The output results.

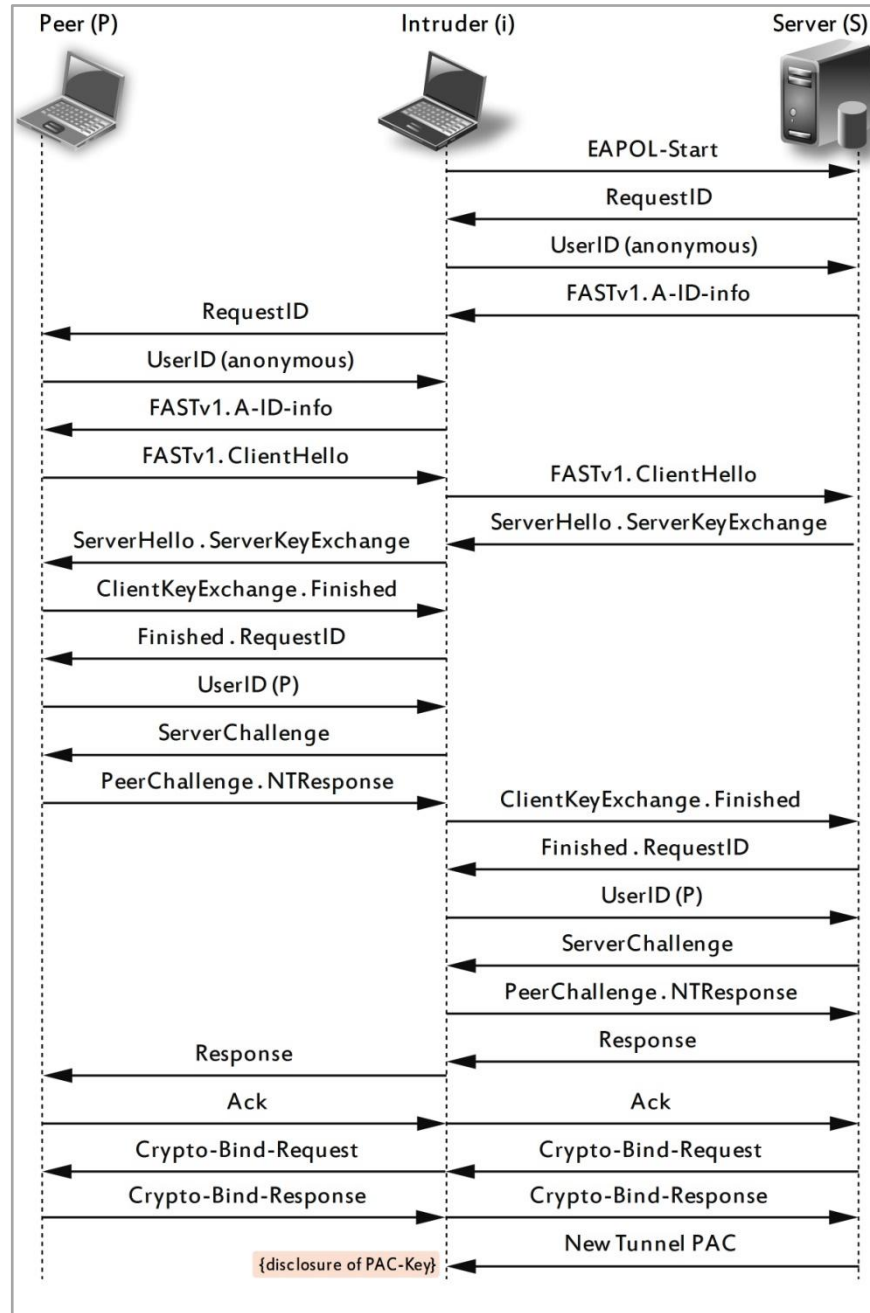


Figure 5.7 Attack trace of server-unauthenticated provisioning mode.

5.2 EAP-FAST Authentication Mechanisms

Authentication using PACs makes the EAP-FAST protocol lightweight. Because the PAC is the kind of shared secret, EAP-FAST protocol uses the symmetric cryptography. In the following subsections we will discuss the validation of EAP-FAST protocol when it uses the Tunnel PAC and User Authorization PAC.

5.2.1 Tunnel Establishment with Tunnel PAC

In this EAP-FAST authentication mechanism, the secure tunnel is established by abbreviated TLS Handshake protocol. The encrypted PAC-Key, which is located in Tunnel PAC (sent in Client Hello message) becomes the pre-master-key (PMS) (Salowey, Zhou, Eronen & Tschofenig, 2008). An A&B notation of the Tunnel PAC usage in establishing secured tunnel is shown in Figure 5.8.

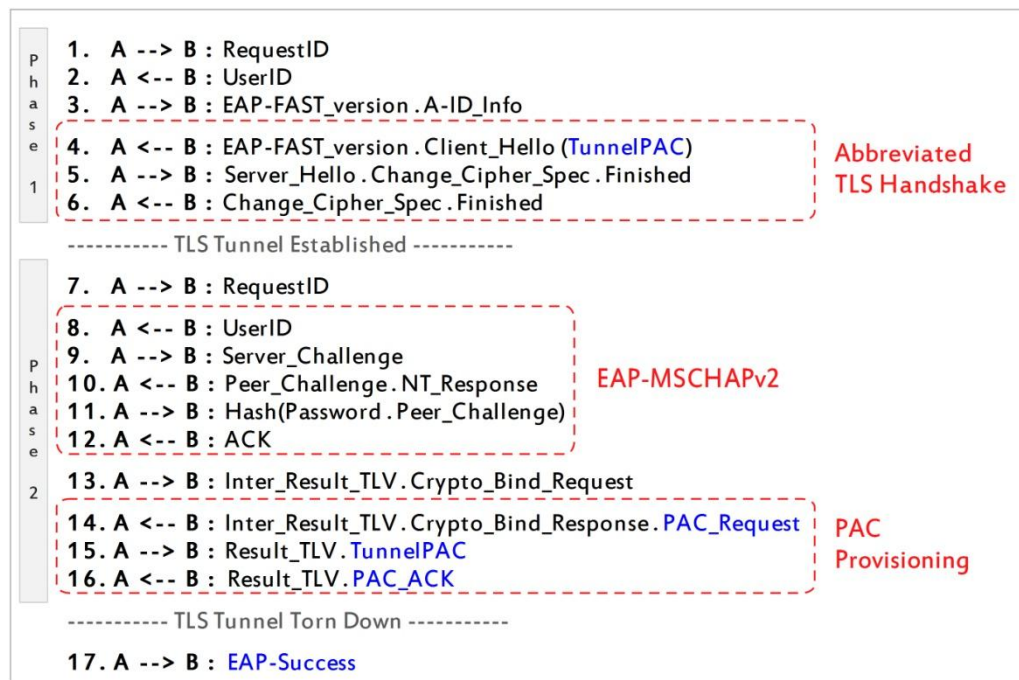


Figure 5.8 Tunnel PAC usage.

When the secure TLS tunnel is established using Tunnel PAC, to avoid aforementioned MiTM attack, it is not necessary to use EAP methods that derive keys. Since the tunnel is established by mutually authenticating the peer and the server using Tunnel PAC, it is possible to use weak EAP methods such EAP-MD5 or EAP-GTC. This property allows the EAP-FAST to be more lightweight.

This protocol mechanism is validated against the goals shown in Figure 5.9 and the output of the validation is shown in Figure 5.10.

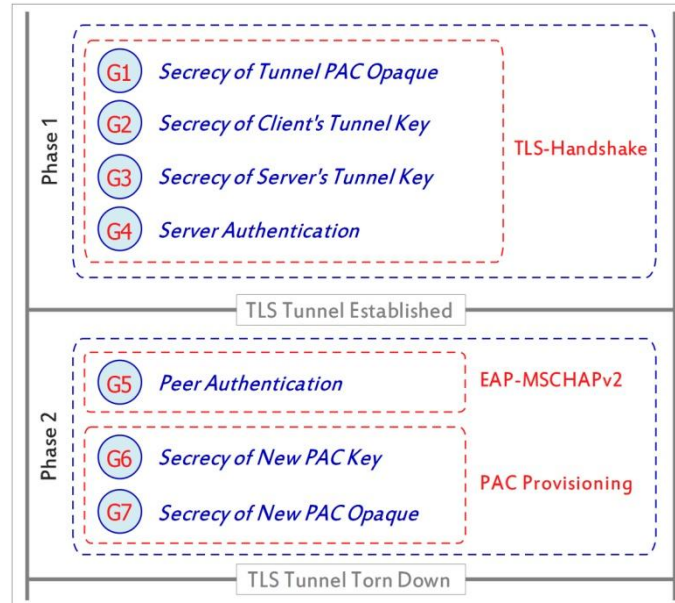


Figure 5.9 Validation goals of EAP-FAST protocol.

```

SUMMARY
SAFE

BACKEND
CL-AtSe

STATISTICS

Analysed   : 26164 states
Reachable  : 6540 states
Translation: 0.72 seconds
Computation: 2.41 seconds

```

Figure 5.10 The output results.

5.2.2 Inner Authentication with User-Authorization PAC

It is possible to skip the inner authentication by using the User-Authorization PAC. The crypto-binding TLVs also will not be exchanged due to the absence of any derived inner keys. It should be noted that, the User Authorization PAC does not include PAC-Key. Thus, it should be bounded to the Tunnel PAC (Cam-Winget, McGrew, Salowey & Zhou, 2009). We bounded it with Tunnel PAC by inserting the hash of the Tunnel PAC into the User Authorization PAC. A Figure 5.11 depicts the A&B notation of the User-Authorization PAC usage within the tunnel.

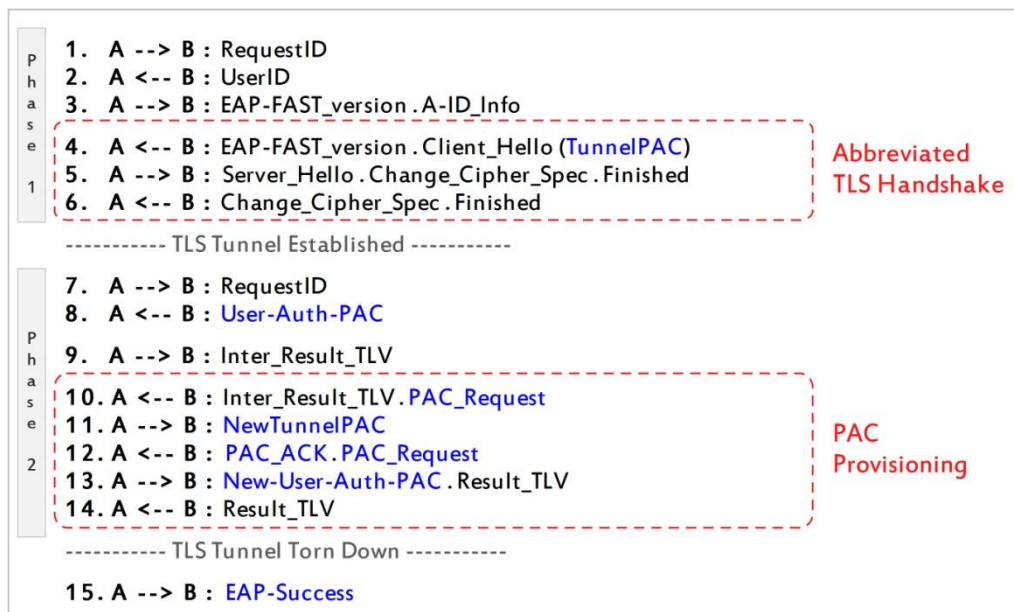


Figure 5.11 User-authorization PAC usage.

This protocol mechanism is validated against the same goals as Tunnel PAC usage mechanism with extra "*secrecy of New User Authentication PAC*" goal in the second phase. The output of the validation is shown in Figure 5.12.

```

SUMMARY
  SAFE

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 39082 states
  Reachable  : 12447 states
  Translation: 0.42 seconds
  Computation: 5.95 seconds
  
```

Figure 5.12 The output results.

5.3 The Four-Way Handshake Protocol

After each successful authentication, the Four-Way handshake process occurs to generate user data encryption keys (PTK) from the seed material derived as the result of the EAP-FAST protocol (IEEE, 2007). An A&B notation of the Four-Way Handshake protocol is depicted in Figure 5.13 and the output shown in Figure 5.14.

1. **A --> B** : ANonce . (SQN) . PMKID
2. **A <-- B** : SNonce . (SQN) . B_RSNIE . MIC1
3. **A --> B** : ANonce . (SQN+1) . A_RSNIE . {GTK}_PTK . MIC2
4. **A <-- B** : (SQN+1) . MIC3

Figure 5.13 The four-way handshake protocol.

where,

A and B : Authenticator and Supplicant respectively

PMK : (Pre-existing) Pairwise Master Key

SQN : Sequence Number

PMKID : Used for Roaming, PMKID = HMAC(PMK.A.B)

PTK : Pairwise Temporal Key = PRF(PMK.A.B.ANonce.SNonce)

GMK : Group Master Key = Randomly generated value

GNonce : Group Nonce generated by Authenticator

GTK : Group Temporal Key = PRF(GMK.A.GNonce)

MIC : Message Integrity Code

MIC1 = MIC(PMK.SNonce. SQN.B_RSNIE)

MIC2 = MIC(PTK.ANonce.(SQN+1).A_RSNIE.{GTK}_PTK)

MIC3 = MIC(PTK.(SQN+1))

```

SUMMARY
  SAFE

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 13 states
  Reachable  : 7 states
  Translation: 0.02 seconds
  Computation: 0.00 seconds

```

Figure 5.14 The output results.

CHAPTER SIX

CONCLUSION

In this research, we discussed authentication, confidentiality and integrity methods of Wireless LAN Security, the RSNA establishment procedures in infrastructure networks, compare the security properties of the widely used TLS-based EAP-methods which are defined in IETF RFCs. We mainly focused on the EAP-FAST protocol because of its attracting security features such as using PACs (shared secrets) to establish a TLS tunnel instead of digital certificates. Using AVISPA model-checker, we validated the different EAP-FAST authentication scenarios and the Four-Way Handshake key management protocol.

Since, manually deploying PACs is not efficient, PACs are typically deployed dynamically. Server-unauthenticated provisioning mode of dynamic PAC deployment doesn't need certificates for PAC distribution. But, based on the results of AVISPA, this provisioning mode is vulnerable to MiTM attack, which couldn't be detected and prevented as stated in Cam-Winget & others (2009). Moreover, this mode is also highly vulnerable to offline-dictionary attack.

According to the output results of AVISPA, EAP-FAST protocol can be SAFE in spite of authentication service when PAC is provisioned in server-authenticated provisioning mode. It means, EAP-FAST is still dependent on at least server-side certificate to provision the wireless clients with valid (and unique) PACs.

Note that, EAP-FAST requires the server certificate only once in the beginning (when the user has not valid PAC) and all subsequent EAP-FAST sessions skip the PAC provisioning. It makes EAP-FAST faster than other certificate-based EAP methods. Thus, EAP-FAST can be the best alternative authentication method in environments where certificate-based methods are already deployed. Furthermore, there is available an EAP-FAST version 2 as an Internet draft which provides an additional security property known as channel binding.

Authors of AVISPA are currently working on the AVANTSSAR project which is the successor of AVISPA. AVANTSSAR supports new versions of most model-checkers those are capable of analyzing including all AVISPA's constraints. For now, AVANTSSAR is not as much popular. As a future work, the EAP-FAST protocol can be analyzed using AVANTSSAR toolset and may be defined new method to distribute PACs to the wireless clients.

REFERENCES

- Aboba, B. & Calhoun, P. (2003). *RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)*. IETF RFC 3579, Informational. Retrieved June 10, 2011, from <http://tools.ietf.org/pdf/rfc3579.pdf>.
- Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., & Levkowetz, H. (2004). *Extensible Authentication Protocol (EAP)*. IETF RFC 3748, Standards Track. Retrieved November 20, 2010, from <http://tools.ietf.org/pdf/rfc3748.pdf>.
- Armando, A. & Compagna, L. (2004). SATMC: A sat-based model checker for security protocols. *Alferes, J.J., Leite, J. (Ed.) JELIA 2004. LNCS (LNAI), vol. 3229, Springer, Heidelberg. 730–733*. Retrieved June 10, 2011, from <http://www.avispa-project.org/papers/satmc-sd-jelia04.ps>.
- Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Heam, P. C., Kouchnarenko, O., Mantovani, J., Modersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Vigano, L., & Vigneron, L. (2005). The AVISPA tool for the automated validation of internet security protocols and Applications. *Etessami, K. and Rajamani S. K. (Ed.). 17th International Conference on Computer Aided Verification, CAV 2005, LNCS 3576 of Lecture Notes in Computer Science, 281–285. Springer*. Retrieved January 13, 2011, from <http://www.avispa-project.org/papers/avispa-cav05.ps>.
- Asokan, N., Niemi, V., & Nyberg, K. (2002). "Man-in-the-middle in tunneled authentication protocols". *IACR ePrint Archive Report 2002/163*. Retrieved June 10, 2011, from <http://eprint.iacr.org/2002/163.pdf>.
- Automated Validation of Internet Security Protocols and Applications (2003a). *Deliverable 2.1: The High-Level Protocol Specification Language*. Retrieved January 13, 2011, from <http://www.avispa-project.org/delivs/2.1/d2-1.pdf>.

Automated Validation of Internet Security Protocols and Applications (2003b).
Deliverable 2.3: The Intermediate Format. Retrieved January 13, 2011, from
<http://www.avispa-project.org/delivs/2.3/d2-3.pdf>.

Automated Validation of Internet Security Protocols and Applications (2003c).
Deliverable 6.1: List of selected problems. Retrieved January 13, 2011, from
<http://www.avispa-project.org/delivs/6.1/d6-1.ps>

Automated Validation of Internet Security Protocols and Applications (2005).
Deliverable 6.2: Specification of the Problems in the High-Level Specification Language. Retrieved December 11, 2010, from <http://www.avispa-project.org/delivs/6.2/d6-2.pdf>.

Automated Validation of Internet Security Protocols and Applications (2006a).
HLPSSL Tutorial. Retrieved December 11, 2010, from <http://www.avispa-project.org/package/tutorial.pdf>.

Automated Validation of Internet Security Protocols and Applications (2006b).
AVISPA v1.1 User Manual. Retrieved November 20, 2010, from
<http://www.avispa-project.org/package/user-manual.pdf>.

Basin, D., Modersheim, S., & Vigano, L. (2005). OFMC: A symbolic model-checker for security protocols. *International Journal of Information Security*, 4(3), 181–208. Retrieved June 10, 2011, from <http://www.avispa-project.org/papers/ofmc-jis05.pdf>.

Boichut, Y., Heam, P.C., Kouchnarenko, O., & Oehl, F. (2004). Improvements on the Genet and Klay technique to automatically verify security protocols. *Proc. Int. Workshop on Automated Verification of Infinite-State Systems (AVIS 2004), joint to ETAPS 2004*, 1–11. Retrieved December 11, 2010, from <http://www.avispa-project.org/papers/boichutheamkouchnarenkoAVIS.ps>.

- Cam-Winget, N., McGrew, D., Salowey, J., & Zhou, H. (2007). *The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)*. IETF RFC 4851, Informational. Retrieved February 8, 2011, from <http://tools.ietf.org/pdf/rfc4851.pdf>.
- Cam-Winget, N., McGrew, D., Salowey, J., & Zhou, H. (2009). *Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)*. IETF RFC 5422, Informational. Retrieved February 8, 2011, from <http://tools.ietf.org/pdf/rfc5422.pdf>.
- Cervesato, I. The Dolev-Yao intruder is the most powerful attacker. *Advanced Engineering and Sciences Division. IIT Industries. Inc.* 2560 Huntington Avenue, Alexandria, VA 22303-1410-USA. Retrieved January 13, 2011, from <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=FFB7BE7DFE78AC3C10FC90FE46C2046B?doi=10.1.1.21.2903&rep=rep1&type=ps>.
- Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P. H., Mantovani, J., Modersheim, S. & Vigneron, L. (2004). A high-level protocol specification language for industrial security-sensitive protocols. *Automated Software Engineering. Proceedings of the Workshop on Specification and Automated Processing of Security Requirements, SAPS 04, Austrian Computer Society*, 193-205. Retrieved January 13, 2011, from <http://www.avispa-project.org/papers/hlpsl-saps04.ps>.
- Coleman, D. D., & Westcott, D. A. (2009). *CWNA: Certified wireless network administrator study guide*. Indiana: Wiley Publishing. Retrieved March 10, 2010, from <http://my.safaribooksonline.com/book/certification/cwna/9780470438909>.
- Coleman, D. D., Westcott, D. A., Harkins, B., & Jackman, S. (2010). *CWSP: Certified wireless security professional official study guide* (1st ed.). Indiana: Wiley Publishing. Retrieved May 10, 2010, from <http://my.safaribooksonline.com/book/certification/cwsp/9780470438916>.

- Dengg, J., Friedl, W., Hortler, P., Jager, M., Lehner, M., Macskasi, C., Matscheko, M., Pumberger, G., Ritt, A. & Wasilewski, M. (2009). *WLAN security & encryption*. Retrieved February 8, 2011, from http://tuxworld.homelinux.org/papers/Wlan_Security_paper.pdf.
- Dierks, T. & Rescorla, E. (2006). *The Transport Layer Security (TLS) Protocol Version 1.1*. IETF RFC 4346, Standards Track. Retrieved February 8, 2011, from <http://tools.ietf.org/pdf/rfc4346.pdf>.
- Dierks, T. & Rescorla, E. (2008). *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF RFC 5246, Standards Track. Retrieved February 8, 2011, from <http://tools.ietf.org/pdf/rfc5246.pdf>.
- Dolev, D., & Yao, A. (1983). On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 198-208. Retrieved January 13, 2011, from <http://www.cs.huji.ac.il/~dolev/pubs/dolev-yao-ieee-01056650.pdf>.
- Funk, P., & Blake-Wilson, S. (2008). *Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)*. IETF RFC 5281, Informational. Retrieved December 11, 2010, from <http://tools.ietf.org/pdf/rfc5281.pdf>.
- Geier, J. (2008). *Implementing 802.1X security solutions for wired and wireless networks*. Indiana: Wiley Publishing. Retrieved February 8, 2011, from <http://www.scribd.com/doc/25919421/Implementing-802-1X-Security-Solutions-for-Wired-and-Wireless-Networks>.
- Glouche, Y., Genet, T. & Houssay, E. (2008). *SPAN: a security protocol animator for AVISPA. Version 1.5 user manual*. INRIA/IRISA. LANDE Project. Retrieved May 20, 2011, from http://www.irisa.fr/celtique/genet/Publications/papier_artist.pdf.

- Haas, H. (2010). *WLAN: Security summary*. Retrieved February 8, 2011, from <https://www.ict.tuwien.ac.at/lva/384.081/modules/slides/B4-WLAN-Sec.pdf>.
- Hoeper, K., & Chen, L. (2007). Where EAP security claims fail. *The 4th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Retrieved December 11, 2010, from <http://eudl.icst.org/pdf/10.4108/icst.qshine.2007.1150>.
- Hoeper, K., & Chen, L. (2009). *Recommendation for EAP Methods Used in Wireless Network Access Authentication*. NIST Special Publication 800-120. Retrieved March 10, 2011, from <http://csrc.nist.gov/publications/nistpubs/800-120/sp800-120.pdf>.
- Institute of Electrical and Electronics Engineers (2004a). *Local and Metropolitan Area Networks: Port-Based Network Access Control*. IEEE Std. 802.1X-2004. Retrieved March 10, 2010, from <http://ieeexplore.ieee.org/iel5/9828/30983/01438730.pdf>.
- Institute of Electrical and Electronics Engineers (2004b). *Supplement to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Specification for Enhanced Security*. IEEE Std. 802.11i-2004. Retrieved December 11, 2010, from <http://ieeexplore.ieee.org/stampPDF/getPDF.jsp?tp=&arnumber=1318903>.
- Institute of Electrical and Electronics Engineers (2007). *Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std. 802.11-2007. Retrieved May 10, 2010, from <http://standards.ieee.org/getieee802/download/802.11-2007.pdf>.

- Microsoft Corporation (2011). *MS-CHAP: Extensible Authentication Protocol Method for Microsoft Challenge Handshake Authentication Protocol (CHAP) Specification*. EAP-MSCHAPv2. Retrieved May 20, 2011, from <http://msdn2.microsoft.com/en-us/library/cc224612.aspx>.
- Modersheim, S., Vigano, L. & von Oheimb, D. (2005). Automated validation of security protocols (AVASP'05). *The 8th European Joint Conference on Theory and Practice of Software (ETAPS 2005)*. Retrieved February 8, 2011, from <http://www.avispa-project.org/avasp/module1and3-avasp05.pdf>.
- Palekar, A., Simon, D., Salowey, J., Zhou, H., Zorn, G., & Josefsson, S. (2004). *Protected EAP Protocol (PEAP) Version 2*. Internet-Draft, Informational. Retrieved December 11, 2010, from <http://tools.ietf.org/pdf/draft-josefsson-pppext-eap-tls-eap-10.pdf>.
- Patel, R., Borisaniya, B., Patel, A., Patel, D., Rajarajan, M., & Zisman, A. (2010). Comparative analysis of formal model checking tools for security protocol verification. *CCIS 89, Springer*, 152–163. Retrieved February 8, 2011, from <http://www.springerlink.com/content/t4hu114351x67524/fulltext.pdf>.
- Salowey, J., Zhou, H., Eronen, P., & Tschofenig, H. (2008). *Transport Layer Security (TLS) Session Resumption without Server-Side State*. IETF RFC 5077, Standards Track. Retrieved May 20, 2011, from <http://tools.ietf.org/pdf/rfc5077.pdf>.
- Simon, D., Aboba, B., & Hurst, R. (2008). *The EAP-TLS Authentication Protocol*. IETF RFC 5216, Standards Track. Retrieved December 11, 2010, from <http://tools.ietf.org/pdf/rfc5216.pdf>.
- Stanley, D., Walker, J., & Aboba, B. (2005). *Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs*. IETF RFC 4017, Informational. Retrieved January 13, 2011, from <http://tools.ietf.org/pdf/rfc4017.pdf>.

- Turuani, M. (2006). The CL-Atse protocol analyser. *Pfenning, F. (Ed.), Proceedings of 17th International Conference on Rewriting Techniques and Applications, RTA, Lecture Notes in Computer Science, Seattle (WA), Springer*. Retrieved May 20, 2011, from <http://www.springerlink.com/content/u24t310898j11612/fulltext.pdf>.
- Vigano, L. (2006). Automated security protocol analysis with the AVISPA tool. *Electronic Notes in Theoretical Computer Science 155*, 61–86. Retrieved December 11, 2010, from <http://www.avispa-project.org/papers/avispa-mfps21.pdf>.
- Zhou, H., Cam-Winget, N., Salowey, J., & Hanna, S. (2011). *Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST) Version 2*. Internet-Draft, Standards Track. (work in progress). Retrieved December 20, 2011, from <http://tools.ietf.org/pdf/draft-ietf-emu-eap-tunnel-method-01.pdf>.
- Zorn, G. (2000). *Microsoft PPP CHAP Extensions, Version 2*. IETF RFC 2759, Informational. Retrieved May 20, 2011, from <http://tools.ietf.org/pdf/rfc2759.pdf>.
- Zorn, G. (2001). *Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE)*. IETF RFC 3079, Informational. Retrieved May 20, 2011, from <http://tools.ietf.org/pdf/rfc3079.pdf>.

APPENDIX A

AVISPA FAQ

The following selected questions were answered by AVISPA Team members in mailing list. We decided to give this in questions and answers form since it is more expressive.

Q1: Why does AVISPA uses terms "SAFE" and "UNSAFE" rather than Secure and either Not-Secure, Non-Secure or Insecure?

A1: A protocol validated with AVISPA is SAFE within the scope of security goals and the analysis scenario given. Since AVISPA is not analyzing all possible executions of the protocol, it cannot be assumed "ABSOLUTELY SECURE".

Q2: Why does AVISPA uses the term "validation" instead of "verification"?

A2: OFMC, CL-AtSe and SATMC back-end analyzers search for attacks, or traces that falsify the goals of the protocol. Thus they do 'FALSIFICATION' (i.e. detection of attacks). Only TA4SP analyzer does 'VERIFICATION' (i.e. proving the protocol is correct). So, AVISPA's goal is both falsification and (bounded) verification, not exclusively one or the other. Hence, VALIDATION.

Q3: Is HLPSL a programming language or not?

A3: HLPSL is a (formal) modeling language but not a programming language. The semantics is based on the Temporal Logic of Actions, so it's closer to a logic.

Q4: Are there any known instances of False Positives (i.e. AVISPA says SAFE, even when there is a known attack in the literature or community) or False Negatives (i.e. AVISPA says UNSAFE, although the attack proposed is not a valid attack)?

A4: It depends on the model and the modeler. The modeler can model the protocol incorrectly. All attacks that are found are really attacks on the model (even if the model is constructed badly), and no attacks on the specified analysis scenario are missed.

Q5: If AVISPA tools detect many attacks, will it show all those attacks?

A5: Only the first attack found is shown. After fixing this attack or removing the respective goal from the goals section, by re-running the tools, any further attack will be shown.

Q6: What is the difference between weak authentication and strong authentication in AVISPA?

A6: Strong authentication allows to check for replay attacks, while weak authentication does not. To search for replay attacks simply put two sessions between a and b in parallel. For instance:

```
role environment()
def=
...
composition
  session(a,b,ka,kb)
  ^ session(a,b,ka,kb)
end role
```

But note that parallel sessions may generate other kinds of attacks.

Q7: Is it possible to model Keyed Hash Functions (HMAC/CMAC) in AVISPA?

A7: Yes, AVISPA supports keyed hash functions. It depends on the key material of the protocol. Simply you can use $H(K,Msg)$, where H is a hash_func, K is a symmetric_key and Msg is the message. If you want to model MACs based on public key signature, you can use $\{Msg\}_{inv(Kp)}$, where Kp is a public_key.

Q8: What is the difference between modeling symmetric encryption and asymmetric encryption in AVISPA?

A8: The syntax for encryption is the same: $\{Msg\}_K$, whatever is the type of K (symmetric key, public key, private key). In AVISPA $private_key = inv(public_key)$. So it is considered that

$\{\{Msg\}_{inv(K)}\}_K$ is equal to Msg . But in the real world signing, encrypting and decrypting messages are done by different algorithms.

Q9: Is it possible to model conditional states (like IF-THEN-ELSE) in AVISPA?

A9: The left-hand side of transitions are conditions, so by writing two transitions, one for each case, both having the same state number it can be modeled. So the analyzer will have to test the two solutions. For instance:

IF Var1 = Var2 then SND(Answer1) else SND(Answer2)

1a. State = 1 \wedge Var1 = Var2 \wedge RCV(Message') \Rightarrow State' = 2 \wedge SND(Answer1)

1b. State = 1 \wedge Var1 \neq Var2 \wedge RCV(Message') \Rightarrow State' = 3 \wedge SND(Answer2)

Q10: Does AVISPA support wireless environment?

A10: AVISPA tools handle only Dolev-Yao channel/intruder. Anyway, Dolev-Yao has most strict requirements, so if the protocol is safe with Dolev-Yao model, you can be sure that it will be safe also for wireless environments.

Q11: Does AVISPA support algebraic equations?

A11: Only CL-AtSe supports XOR and exponentiation, while OFMC supports only exponentiation. The other back-ends do not support these properties. In addition OFMC is capable of handling user defined algebraic expressions (equations). (Now, new versions of all back-ends with new features are already available)

Q12: Does AVISPA support unbounded sessions?

A12: TA4SP can handle unbounded number of sessions on some restricted classes of protocols and for secrecy only. The other back-ends do not support it.

Q13: Many protocols assume that the server is trusted and not compromised. How can I model this assumption in AVISPA?

A13: Specifying sessions; depends on the protocol and what sorts of attacks the protocol might be vulnerable to. For instance, if a protocol between A and B is vulnerability to Man-in-the-middle attacks, sessions between (A,B) (A,I) and (I,B) are good bets. In case where the server is assumed trusted; there is no need to analyze a session where the intruder plays the role of the server.

APPENDIX B

EAP-FAST Authentication Mechanism

B.1 Tunnel PAC Usage

Phase 1: TLS Tunnel Establishment using Tunnel PAC.

Phase 2: EAP-MSCHAPv2 method and new Tunnel PAC Provisioning.

B.1.1 The HLPSL Specification

```

role server(
    S, P      : agent,
    Password  : symmetric_key,
    PACInfo   : text,
    Kserver   : symmetric_key,
    Hash1     : hash_func,
    Hash2     : hash_func,
    PRF       : hash_func,
    KEYGEN    : hash_func,
    SND, RCV  : channel (dy) )
played_by S def=

local
    State      : nat,
    Snonce     : text,
    Pnonce     : text,
    SessionID  : text,
    Csuite     : text,
    CCspec     : text,
    MS         : hash(symmetric_key.text.text),
    Finished   : hash(hash(symmetric_key.text.text).agent.agent.text.text.text),
    ClientKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
    ServerKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
    UserID     : text,
    FASTv1    : text,
    TLSv1     : text,
    CsuiteList : text,
    Schallenge : text,
    Pchallenge : text,
    NTResponse : hash(symmetric_key.text.text.agent),
    Inter_result_tlv : text,
    Bind_version : text,
    CMKnonce   : text,
    MasterKey   : hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent)),
    MasterSendKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    MasterReceiveKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    MSK        : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    Seed       : hash(hash(symmetric_key.text.text).text.text.text),
    CMK        : hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    Compound_MAC1 : hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    Compound_MAC2 : hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    Crypto_bind_request : text.text.text.text.
                hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    Crypto_bind_response : text.text.text.text.
                hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    PAC_type      : text,
    Result_tlv    : text,
    PAC_key       : symmetric_key,
    PAC_lifetime  : text,
    A_ID         : text,
    PAC_info      : text.text.text.text,
    PAC_enc_key   : symmetric_key,

```

```

PAC_opaque      : {symmetric_key.text.text.text}_symmetric_key,
PAC_ack         : text,
PACkey          : symmetric_key

init State := 0

transition

1. State = 0 /\ RCV(start) =|>
   State' := 2 /\ SND(eap_request_id)

2. State = 2 /\ RCV(UserID') =|>
   State' := 4 /\ FASTv1' := new()
              /\ SND(start_fast.FASTv1'.a_id_info)

3. State = 4 /\ RCV(FASTv1.TLSv1'.session_id_0.Pnonce'.CsuiteList'.{PACkey'.PACinfo}_Kserver)
   =|>
   State' := 6 /\ Snonce'      := new()
              /\ SessionID' := new()
              /\ Csuite'     := new()
              /\ CCspec'    := new()
              /\ MS'        := PRF(PACkey'.Snonce'.Pnonce')
              /\ Finished'  := Hash1(MS'.P.S.Pnonce'.Csuite'.SessionID')
              /\ ClientKEY' := KEYGEN(P.Pnonce'.Snonce'.MS')
              /\ ServerKEY' := KEYGEN(S.Pnonce'.Snonce'.MS')
              /\ SND(TLSv1'.Snonce'.SessionID'.Csuite'.CCspec'.{Finished'}_ServerKEY')
              /\ witness(S,P,nonces,Pnonce'.Snonce')
              /\ secret(PACkey',sec_pac,{S,P})

##### Phase 2 #####

4. State = 6 /\ RCV(CCspec.{Finished}_ClientKEY) =|>
   State' := 8 /\ SND({eap_request_id}_ServerKEY)

5. State = 8 /\ RCV({P}_ClientKEY) =|>
   State' := 10 /\ Schallenge' := new()
               /\ SND({Schallenge'}_ServerKEY)

6. State = 10 /\ RCV({Pchallenge'.NTResponse'}_ClientKEY)
   /\ NTResponse' = Hash2(Password.Pchallenge'.Schallenge.P) =|>
   State' := 12 /\ SND({Hash2(Password.Pchallenge')}_ServerKEY)
               /\ request(S,P,peer_proof,NTResponse')

7. State = 12 /\ RCV({auth_ack}_ClientKEY) =|>
   State' := 14 /\ Inter_result_tlv' := new()      % success
               /\ Bind_version'    := new()      % same version
               /\ CMKnonce'       := new()
               /\ MasterKey'      := Hash2(Hash2(Hash2(Password)).NTResponse)
               /\ MasterSendKey' := PRF(MasterKey')
               /\ MasterReceiveKey' := KEYGEN(MasterKey')
               /\ MSK'           := MasterReceiveKey'.MasterSendKey'
               /\ Seed'          := PRF(MS.seed_label.Pnonce'.Snonce)
               /\ CMK'           := PRF(Seed'.cmk_label.MSK')
               /\ Compound_MAC1' := Hash2(CMK'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1')
               /\ Crypto_bind_request' := Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'
   /\ SND({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)

##### PAC Provisioning #####

8. State = 14
   /\ RCV({Inter_result_tlv.Bind_version.FASTv1.one.CMKnonce.Compound_MAC2'.PAC_type'}_ClientKEY)
   /\ Compound_MAC2' = Hash2(CMK.Bind_version.FASTv1.one.CMKnonce) =|>
   State' := 16 /\ Result_tlv' := new()
               /\ PAC_key'    := new()
               /\ PAC_lifetime := new()
               /\ A_ID'      := new()
               /\ PAC_info'  := PAC_lifetime'.A_ID'.a_id_info.PAC_type'
               /\ PAC_enc_key' := new()
               /\ PAC_opaque' := {PAC_key'.PAC_info'}_PAC_enc_key'
               /\ SND({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
               /\ secret(PAC_key',sec_packey,{S,P})
               /\ secret(PAC_opaque',sec_pacopaque,{S,P})

9. State = 16 /\ RCV({Result_tlv.PAC_ack'}_ClientKEY) =|>
   State' := 18 /\ SND(eap_success)

end role

#####

role peer(
  S, P      : agent,
  Password : symmetric_key,
  PACkey   : symmetric_key,

```

```

    PACInfo : text,
    Ticket  : {symmetric_key.text}_symmetric_key,
    Hash1   : hash_func,
    Hash2   : hash_func,
    PRF     : hash_func,
    KEYGEN  : hash_func,
    SND, RCV : channel (dy) )
played_by P def=

local
  State      : nat,
  Snonce     : text,
  Pnonce     : text,
  SessionID  : text,
  Csuite     : text,
  CCSpec     : text,
  MS         : hash(symmetric_key.text.text),
  Finished   : hash(hash(symmetric_key.text.text).agent.agent.text.text.text),
  ClientKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
  ServerKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
  UserID     : text,
  FASTv1    : text,
  TLSv1     : text,
  CsuiteList: text,
  Schallenge : text,
  Pchallenge : text,
  NTResponse: hash(symmetric_key.text.text.agent),
  Inter_result_tlv : text,
  Bind_version : text,
  CMKnonce   : text,
  MasterKey  : hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent)),
  MasterSendKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
  MasterReceiveKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
  MSK        : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
  Seed       : hash(hash(symmetric_key.text.text).text.text.text),
  CMK        : hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                    hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                    hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
  Compound_MAC1 : hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))).
                    .text.text.text.text ),
  Compound_MAC2 : hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))).
                    .text.text.text.text ),
  Crypto_bind_request : text.text.text.text.
                    hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))).
                    .text.text.text.text ),
  Crypto_bind_response: text.text.text.text.
                    hash( hash( hash(hash(symmetric_key.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
                              hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))).
                    .text.text.text.text ),
  PAC_type      : text,
  Result_tlv    : text,
  PAC_key       : symmetric_key,
  PAC_lifetime  : text,
  A_ID         : text,
  PAC_info      : text.text.text.text,
  PAC_opaque    : {symmetric_key.text.text.text.text}_symmetric_key,
  PAC_ack       : text

init State := 1
transition

1. State = 1 /\ RCV(eap_request_id) =|>
   State' := 3 /\ UserID' := new()
           /\ SND(UserID')

2. State = 3 /\ RCV(start_fast.FASTv1'.a_id_info) =|>
   State' := 5 /\ TLSv1' := new()
               /\ Pnonce' := new()
               /\ CsuiteList' := new()
               /\ SND(FASTv1'.TLSv1'.session_id_0.Pnonce'.CsuiteList'.Ticket)

3. State = 5 /\ RCV(TLSv1.Snonce'.SessionID'.Csuite'.CCSpec'.{Finished'}_ServerKEY')
   /\ Finished' = Hash1(PACkey.Snonce'.Pnonce'.P.S.Pnonce.Csuite'.SessionID')
   /\ ServerKEY' = KEYGEN(S.Pnonce.Snonce'.PRF(PACkey.Snonce'.Pnonce)) =|>
   State' := 7 /\ MS' := PRF(PACkey.Snonce'.Pnonce)
               /\ ClientKEY' := KEYGEN(P.Pnonce.Snonce'.MS')

```

```

        /\ SND(CCspec'.{Finished'}_ClientKEY')

##### Phase 2 #####
4. State = 7 /\ RCV({eap_request_id}_ServerKEY) =|>
   State' := 9 /\ SND({P}_ClientKEY)
              /\ secret(ClientKEY,sec_clientkey,{P,S})
              /\ secret(ServerKEY,sec_serverkey,{P,S})
              /\ request(P,S,nonces,Pnonce.Snonce)

5. State = 9 /\ RCV({Schallenge'}_ServerKEY) =|>
   State' := 11 /\ Pchallenge' := new()
                /\ NTResponse' := Hash2(Password.Pchallenge'.Schallenge'.P)
                /\ SND({Pchallenge'.NTResponse'}_ClientKEY)
                /\ witness(P,S,peer_proof,NTResponse')

6. State = 11 /\ RCV({Hash2(Password.Pchallenge)}_ServerKEY) =|>
   State' := 13 /\ SND({auth_ack}_ClientKEY)
                /\ MasterKey' := Hash2(Hash2(Hash2(Password)).NTResponse)
                /\ MasterSendKey' := PRF(MasterKey')
                /\ MasterReceiveKey' := KEYGEN(MasterKey')
                /\ MSK' := MasterReceiveKey'.MasterSendKey'
                /\ Seed' := PRF(MS.seed_label.Pnonce.Snonce)
                /\ CMK' := PRF(Seed'.cmk_label.MSK')

##### PAC Provisioning #####
7. State = 13
   /\ RCV({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)
   State' := 15 /\ Compound_MAC1' = Hash2(CMK.Bind_version'.FASTv1.one.CMKnonce') =|>
                /\ Compound_MAC2' := Hash2(CMK.Bind_version'.FASTv1.one.CMKnonce')
                /\ Crypto_bind_response' := Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'
                /\ PAC_type' := new()
/\SND({Inter_result_tlv'.Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'.PAC_type'}_ClientKEY)

8. State = 15 /\ RCV({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
   State' := 17 /\ PAC_info' = PAC_lifetime'.A_ID'.a_id_info.PAC_type' =|>
                /\ PAC_ack' := new()
                /\ SND({Result_tlv'.PAC_ack'}_ClientKEY)

9. State = 17 /\ RCV(eap_success) =|>
   State' := 19

end role

#####

role session(
    S, P : agent,
    Password : symmetric_key,
    PACkey : symmetric_key,
    PACinfo : text,
    Kserver : symmetric_key,
    Ticket : {symmetric_key.text}_symmetric_key,
    Hash1 : hash_func,
    Hash2 : hash_func,
    PRF : hash_func,
    KEYGEN : hash_func )
def=

local PSND,PRCV,SSND,SRCV : channel (dy)
composition
    server (S,P,Password, PACinfo, Kserver, Hash1,Hash2,PRF,KEYGEN,SSND,SRCV)
    /\ peer (S,P,Password,PACkey,PACinfo, Ticket, Hash1,Hash2,PRF,KEYGEN,PSND,PRCV)

end role

role environment() def=

const
    eap_request_id, start_fast: text,
    auth_ack, eap_success : text,
    a_id_info : text,
    session_id_0 : text, % SessionID = 0
    seed_label, cmk_label : text,
    zero, one : text,

    sec_pac,
    sec_packey, sec_pacopaque,
    nonces, peer_proof,
    sec_clientkey,
    sec_serverkey : protocol_id,
    s,p,i : agent,
    kps,kis,kpi : symmetric_key,

```

```

    packey, pacpi, pacsi      : symmetric_key,
    pacinfo                  : text,
    kserver, ksi             : symmetric_key,
    hash1,hash2              : hash_func,
    prf                      : hash_func,
    keygen                   : hash_func

    intruder_knowledge = { s,p,hash1,hash2,prf,keygen,kpi,kis,pacpi,pacsi,ksi }    % pacinfo

    composition
      session(s,p,kps, packey,pacinfo, kserver,{packey.pacinfo}_kserver, hash1,hash2,prf,keygen)
/\   session(s,i,kis, pacpi, pacinfo, kserver,{pacpi.pacinfo}_kserver, hash1,hash2,prf,keygen)
/\   session(i,p,kpi, packey,pacinfo, ksi,      {pacsi.pacinfo}_ksi,      hash1,hash2,prf,keygen)

end role

goal

    secrecy_of sec_pac
    secrecy_of sec_clientkey, sec_serverkey
    secrecy_of sec_packey, sec_pacopaque

    authentication_on nonces          % server authentication
    authentication_on peer_proof     % peer authentication

end goal

environment()

```

B.1.2 The Output Results

```

root@ebakyt-laptop:/avispa# avispa Tunnel-establish-pac.hlp1 --cl-atse

SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Tunnel-establish-pac.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 26164 states
  Reachable  : 6540 states
  Translation: 0.72 seconds
  Computation: 2.41 seconds

root@ebakyt-laptop:/avispa# avispa Tunnel-establish-pac.hlp1 --typed_model=no --cl-atse

SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  UNTYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Tunnel-establish-pac.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS
  Analysed   : 26164 states
  Reachable  : 6540 states
  Translation: 0.71 seconds
  Computation: 2.39 seconds

```

B.2 User Authorization PAC Usage

Phase 1: TLS Tunnel Establishment using Tunnel PAC.

Phase 2: Authenticate user with User Authorization PAC, new Tunnel PAC and new User Authorization PAC Provisioning.

B.2.1 The HLPSL Specification

```

role server(
    S, P      : agent,
    Password  : symmetric_key,
    PACinfo   : text,
    Kserver   : symmetric_key,
    Hash1     : hash_func,
    Hash2     : hash_func,
    PRF       : hash_func,
    KEYGEN    : hash_func,
    SND, RCV  : channel (dy) )

played_by S def=

local
    State      : nat,
    Snonce     : text,
    Pnonce     : text,
    SessionID  : text,
    Csuite     : text,
    CCspec     : text,
    MS         : hash(symmetric_key.text.text),
    Finished   : hash(hash(symmetric_key.text.text).agent.agent.text.text.text),
    ClientKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
    ServerKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
    UserID     : text,
    FASTv1    : text,
    TLSv1     : text,
    CsuiteList : text,
    Inter_result_tlv : text,
    PAC_type   : text,
    PAC_type3  : text,
    Result_tlv : text,
    PAC_key    : symmetric_key,
    PAC_lifetime : text,
    A_ID       : text,
    PAC_info   : text.text.text.text,
    PAC_enc_key : symmetric_key,
    PAC_opaque : {symmetric_key.text.text.text.text}_symmetric_key,
    PAC_ack    : text,
    PACkey     : symmetric_key,
    UserAuthID : text,
    NUserAuthID : text,
    TunnelPAC  : hash({symmetric_key.text}_symmetric_key),
    UserPAC    : {hash({symmetric_key.text}_symmetric_key).text}_symmetric_key,
    NewUserPAC : {hash({symmetric_key.text.text.text.text}_symmetric_key).text}_symmetric_key

init State := 0

transition

1. State = 0 /\ RCV(start) =|>
   State' := 2 /\ SND(eap_request_id)

2. State = 2 /\ RCV(UserID') =|>
   State' := 4 /\ FASTv1' := new()
            /\ SND(start_fast.FASTv1'.a_id_info)

3. State = 4
   /\ RCV(FASTv1.TLSv1'.session_id_0.Pnonce'.CsuiteList'.{PACkey'.PACinfo}_Kserver) =|>
   State' := 6 /\ TunnelPAC' := hash1({PACkey'.PACinfo}_Kserver)
              /\ Snonce'    := new()
              /\ SessionID' := new()
              /\ Csuite'    := new()
              /\ CCspec'    := new()
              /\ MS'        := PRF(PACkey'.Snonce'.Pnonce')
              /\ Finished'  := Hash1(MS'.P.S.Pnonce'.Csuite'.SessionID')
              /\ ClientKEY' := KEYGEN(P.Pnonce'.Snonce'.MS')
              /\ ServerKEY' := KEYGEN(S.Pnonce'.Snonce'.MS')
              /\ SND(TLSv1'.Snonce'.SessionID'.Csuite'.CCspec'.{Finished'}_ServerKEY')
              /\ witness(S,P,nonces,Pnonce'.Snonce')
              /\ secret(PACkey',sec_pac,{S,P})

***** Phase 2: User Authorization PAC usage *****

4. State = 6 /\ RCV(CCspec.{Finished}_ClientKEY) =|>

```

```

State' := 8 /\ SND({eap_request_id}_ServerKEY)

5. State = 8 /\ RCV({UserPAC'}_ClientKEY)
   /\ UserPAC' = {TunnelPAC.UserAuthID'}_Kserver =|>
State' := 10 /\ Inter_result_tlv' := new()
   /\ SND({Inter_result_tlv'}_ServerKEY)
   /\ secret(UserAuthID', sec_useridentity, {S,P})

##### PAC Provisioning #####

6. State = 10 /\ RCV({Inter_result_tlv.PAC_type'}_ClientKEY) =|>
   State' := 12 /\ PAC_key' := new()
   /\ PAC_lifetime' := new()
   /\ A_ID' := new()
   /\ PAC_info' := PAC_lifetime'.A_ID'.a_id_info.PAC_type'
   /\ PAC_enc_key' := new()
   /\ PAC_opaque' := {PAC_key'.PAC_info'}_PAC_enc_key'
   /\ SND({PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
   /\ secret(PAC_key', sec_packey, {S,P})
   /\ secret(PAC_opaque', sec_pacopaque, {S,P})

7. State = 12 /\ RCV({PAC_ack'.PAC_type3'}_ClientKEY) =|>
   State' := 14 /\ Result_tlv' := new()
   /\ NUserAuthID' := new()
   /\ NewUserPAC' := {hash1(PAC_opaque).NUserAuthID'}_PAC_enc_key
   /\ SND({Result_tlv'.NewUserPAC'}_ServerKEY)
   /\ secret(NUserAuthID', sec_userauthid, {S,P})
   /\ secret(NewUserPAC', sec_newuserpac, {S,P})
   /\ request(S,P,peerauth,UserPAC)

8. State = 14 /\ RCV({Result_tlv}_ClientKEY) =|>
   State' := 16 /\ SND(eap_success)

end role

#####

role peer(
  S, P      : agent,
  Password  : symmetric_key,
  PACkey    : symmetric_key,
  PACinfo   : text,
  UserAuthID : text,
  Ticket    : {symmetric_key.text}_symmetric_key,
  UserPAC   : {hash({symmetric_key.text}_symmetric_key).text}_symmetric_key,
  Hash1     : hash_func,
  Hash2     : hash_func,
  PRF       : hash_func,
  KEYGEN    : hash_func,
  SND, RCV  : channel (dy) )
played_by P def=

local
  State      : nat,
  Snonce     : text,
  Pnonce     : text,
  SessionID  : text,
  Csuite     : text,
  CSpec      : text,
  MS         : hash(symmetric_key.text.text),
  Finished   : hash(hash(symmetric_key.text.text).agent.agent.text.text.text),
  ClientKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
  ServerKEY  : hash(agent.text.text.hash(symmetric_key.text.text)),
  UserID     : text,
  FASTlv1   : text,
  TLSv1     : text,
  CsuiteList : text,
  Inter_result_tlv : text,
  PAC_type   : text,
  PAC_type3  : text,
  Result_tlv : text,
  PAC_key    : symmetric_key,
  PAC_lifetime : text,
  A_ID       : text,
  PAC_info   : text.text.text.text,
  PAC_opaque : {symmetric_key.text.text.text.text}_symmetric_key,
  PAC_ack    : text,
  NewUserPAC : {hash({symmetric_key.text.text.text.text}_symmetric_key).text}_symmetric_key

init State := 1

transition

1. State = 1 /\ RCV(eap_request_id) =|>

```

```

State' := 3 /\ UserID' := new()
          /\ SND(UserID')

2. State = 3 /\ RCV(start_fast.FASTv1'.a_id_info) =|>
State' := 5 /\ TLSv1' := new()
          /\ Pnonce' := new()
          /\ CsuiteList' := new()
          /\ SND(FASTv1'.TLSv1'.session_id_0.Pnonce'.CsuiteList'.Ticket)

3. State = 5 /\ RCV(TLSv1'.Snonce'.SessionID'.Csuite'.CCspec'.{Finished'}_ServerKEY')
          /\ Finished' = Hash1(PRF(PACkey.Snonce'.Pnonce).P.S.Pnonce.Csuite'.SessionID')
          /\ ServerKEY' = KEYGEN(S.Pnonce.Snonce'.PRF(PACkey.Snonce'.Pnonce)) =|>
State' := 7 /\ MS' := PRF(PACkey.Snonce'.Pnonce)
          /\ ClientKEY' := KEYGEN(P.Pnonce.Snonce'.MS')
          /\ SND(CCspec'.{Finished'}_ClientKEY')

##### Phase 2: User Authorization PAC usage #####

4. State = 7 /\ RCV({eap_request_id}_ServerKEY) =|>
State' := 9 /\ SND({UserPAC}_ClientKEY)
          /\ secret(ClientKEY, sec_clientkey, {P,S})
          /\ secret(ServerKEY, sec_serverkey, {P,S})
          /\ request(P,S,nonces,Pnonce.Snonce)
          /\ witness(P,S,peerauth,UserPAC)

##### PAC Provisioning #####

5. State = 9 /\ RCV({Inter_result_tlv'}_ServerKEY) =|>
State' := 11 /\ PAC_type' := new() % = '1' for Tunnel PAC
          /\ SND({Inter_result_tlv'.PAC_type'}_ClientKEY)

6. State = 11 /\ RCV({PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
          /\ PAC_info' = PAC_lifetime'.A_ID'.a_id_info.PAC_type =|>
State' := 13 /\ PAC_type3' := new() % = '3' for User Auth. PAC
          /\ PAC_ack' := new()
          /\ SND({PAC_ack'.PAC_type3'}_ClientKEY)

7. State = 13 /\ RCV({Result_tlv'.NewUserPAC'}_ServerKEY) =|>
State' := 15 /\ SND({Result_tlv'}_ClientKEY)

8. State = 15 /\ RCV(eap_success) =|>
State' := 17

end role

#####

role session(
  S, P      : agent,
  Password  : symmetric_key,
  PACkey    : symmetric_key,
  PACinfo   : text,
  Kserver   : symmetric_key,
  UserAuthID : text,
  Ticket    : {symmetric_key.text}_symmetric_key,
  UserPAC   : {hash({symmetric_key.text}_symmetric_key).text}_symmetric_key,
  Hash1     : hash_func,
  Hash2     : hash_func,
  PRF       : hash_func,
  KEYGEN    : hash_func )

def=

  local PSND, PRCV, SSND, SRCV : channel (dy)

  composition
  server(S,P,Password, PACinfo, Kserver, Hash1,Hash2,PRF,KEYGEN,SSND, SRCV)
  /\ peer(S,P,Password,PACkey,PACinfo,UserAuthID,Ticket,UserPAC,Hash1,Hash2,PRF,KEYGEN,PSND,PRCV)

end role

role environment() def=
  const
  eap_request_id, start_fast: text,
  auth_ack, eap_success : text,
  a_id_info : text,
  session_id_0 : text, % SessionID = 0
  seed_label, cmk_label : text,
  zero, one : text,

  sec_pac, sec_useridentity,
  sec_packey, sec_pacopaque,
  nonces, sec_userauthid,
  sec_newuserpac,
  peerauth,

```



```

    sec_clientkey,
    sec_serverkey      : protocol_id,
    s,p,i              : agent,
    kps,kis,kpi        : symmetric_key,
    packey, pacpi, pacsi : symmetric_key,
    pacinfo            : text,
    userauthid         : text,
    kserver, ksi       : symmetric_key,
    hash1,hash2        : hash_func,
    prf                 : hash_func,
    keygen             : hash_func

intruder_knowledge = { s,p,hash1,hash2,prf,keygen,kpi,kis,pacpi,pacsi,ksi }
composition
  session(s,p,kps,packey,pacinfo,kserver,userauthid,{packey.pacinfo}_kserver,
    {hash1({packey.pacinfo}_kserver).userauthid}_kserver,hash1,hash2,prf,keygen)
/\ session(s,i,kis, pacpi, pacinfo,kserver,userauthid,{pacpi.pacinfo}_kserver,
  {hash1({pacpi.pacinfo}_kserver).userauthid}_kserver, hash1,hash2,prf,keygen)
/\ session(i,p,kpi, packey,pacinfo,ksi, userauthid,{pacsi.pacinfo}_ksi,
  {hash1({pacsi.pacinfo}_ksi).userauthid}_ksi, hash1,hash2,prf,keygen)
end role

goal
  secrecy_of sec_pac
  secrecy_of sec_clientkey, sec_serverkey
  secrecy_of sec_useridentity
  secrecy_of sec_packey, sec_pacopaque
  secrecy_of sec_userauthid
  secrecy_of sec_newuserpac
  authentication_on nonces           % server authentication
  authentication_on peerauth         % peer authentication
end goal

environment()

```

B.2.2 The Output Results

```
root@ebakyt-laptop:/avispa# avispa Tunnel-and-user-pac.hlp1 --cl-atse
```

SUMMARY

SAFE

DETAILS

BOUNDED_NUMBER_OF_SESSIONS
TYPED_MODEL

PROTOCOL

/avispa/avispa-1.1//testsuite/results/Tunnel-and-user-pac.if

GOAL

As Specified

BACKEND

CL-AtSe

STATISTICS

Analysed : 39082 states
Reachable : 12447 states
Translation: 0.42 seconds
Computation: 5.95 seconds

```
root@ebakyt-laptop:/avispa# avispa Tunnel-and-user-pac.hlp1 --typed_model=no --cl-atse
```

SUMMARY

SAFE

DETAILS

BOUNDED_NUMBER_OF_SESSIONS
UNTYPED_MODEL

PROTOCOL

/avispa/avispa-1.1//testsuite/results/Tunnel-and-user-pac.if

GOAL

As Specified

BACKEND

CL-AtSe

STATISTICS

Analysed : 46655 states
Reachable : 15541 states
Translation: 0.42 seconds
Computation: 28.31 seconds

APPENDIX C

EAP-FAST Dynamic Provisioning Mechanism

C.1 Server-Authenticated Provisioning

Phase 1: TLS Tunnel Establishment using Server Certificates.

Phase 2: EAP-MSCHAPv2 method and Tunnel PAC Provisioning.

C.1.1 The HLPSL Specification

```

role server(
    S, P      : agent,
    Password  : symmetric_key,
    Kserver   : public_key,
    Kca       : public_key,
    Hash1     : hash_func,
    Hash2     : hash_func,
    PRF       : hash_func,
    KEYGEN    : hash_func,
    SND, RCV  : channel (dy) )
played_by S def=

local
    State      : nat,
    Snonce     : text,
    Pnonce     : text,
    SessionID  : text,
    Csuite     : text,
    PMS        : text,
    CCspec     : text,
    MS         : hash(text.text.text),
    Finished   : hash(hash(text.text.text).agent.agent.text.text.text),
    ClientKEY  : hash(agent.text.text.hash(text.text.text)),
    ServerKEY  : hash(agent.text.text.hash(text.text.text)),
    UserID     : text,
    FASTv1    : text,
    TLSv1     : text,
    CsuiteList : text,
    SHelloDone : text,
    Schallenge : text,
    Pchallenge : text,
    NTResponse : hash(symmetric_key.text.text.agent),
    Inter_result_tlv : text,
    Bind_version : text,
    CMKnonce   : text,
    MasterKey   : hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent)),
    MasterSendKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    MasterReceiveKey : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    MSK        : hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
    Seed       : hash(hash(text.text.text).text.text.text),
    CMK        : hash(hash(hash(text.text.text).text.text.text).text.
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))),
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))),
    Compound_MAC1 : hash(hash(hash(hash(text.text.text).text.text.text).text.
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))
        .text.text.text.text ),
    Compound_MAC2 : hash(hash(hash(hash(text.text.text).text.text.text).text.
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))
        .text.text.text.text ),
    Crypto_bind_request : text.text.text.text.
        hash(hash(hash(hash(text.text.text).text.text.text).text.text.text).text.
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))
        .text.text.text.text ),
    Crypto_bind_response : text.text.text.text.
        hash(hash(hash(hash(text.text.text).text.text.text).text.text.text).text.
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))).
        hash(hash(hash(hash(symmetric_key)).hash(symmetric_key.text.text.agent))))
        .text.text.text.text ),
    PAC_type   : text,
    Result_tlv : text,
    PAC_key    : symmetric_key,
    PAC_lifetime : text,
    A_ID       : text,

```

```

PAC_info      : text.text.text.text,
PAC_enc_key   : symmetric_key,
PAC_opaque    : {symmetric_key.text.text.text.text}_symmetric_key,
PAC_ack       : text

init State := 0

transition

1. State = 0 /\ RCV(start) =|>
   State' := 2 /\ SND(eap_request_id)

2. State = 2 /\ RCV(UserID') =|>
   State' := 4 /\ FASTv1' := new()
              /\ SND(start_fast.FASTv1'.a_id_info)

3. State = 4 /\ RCV(FASTv1.TLSv1'.session_id_0.Pnonce'.CsuiteList') =|>
   State' := 6 /\ Snonce' := new()
              /\ SessionID' := new()
              /\ Csuite' := new()
              /\ SHelloDone' := new()
              /\ SND(TLSv1'.Snonce'.SessionID'.Csuite'.{S.Kserver}_inv(Kca).SHelloDone')
              /\ witness(S,P,nonces,Pnonce'.Snonce')

4. State = 6 /\ RCV({PMS'}_Kserver.CCspec'.{Finished'}_ClientKEY')
   /\ Finished' = Hash1(PRF(PMS'.Pnonce.Snonce).P.S.Pnonce.Csuite.SessionID)
   /\ ClientKEY' = KEYGEN(P.Pnonce.Snonce.PRF(PMS'.Pnonce.Snonce)) =|>
   State' := 8 /\ MS' := PRF(PMS'.Pnonce.Snonce)
              /\ ServerKEY' := KEYGEN(S.Pnonce.Snonce.MS')
              /\ SND(CCspec'.{Finished'}.eap_request_id)_ServerKEY')

##### Phase 2 #####

5. State = 8 /\ RCV({P}_ClientKEY) =|>
   State' := 10 /\ Schallenge' := new()
               /\ SND({Schallenge'}_ServerKEY)

6. State = 10 /\ RCV({Pchallenge'.NTResponse'}_ClientKEY)
   /\ NTResponse' = Hash2(Password.Pchallenge'.Schallenge.P) =|>
   State' := 12 /\ SND({Hash2(Password.Pchallenge')}_ServerKEY)
               /\ request(S,P,peer_proof,NTResponse')

7. State = 12 /\ RCV({auth_ack}_ClientKEY) =|>
   State' := 14 /\ Inter_result_tlv' := new() % success
               /\ Bind_version' := new() % same version
               /\ CMKnonce' := new()
               /\ MasterKey' := Hash2(Hash2(Hash2(Password)).NTResponse)
               /\ MasterSendKey' := PRF(MasterKey')
               /\ MasterReceiveKey' := KEYGEN(MasterKey')
               /\ MSK' := MasterReceiveKey'.MasterSendKey'
               /\ Seed' := PRF(MS.seed_label.Pnonce.Snonce)
               /\ CMK' := PRF(Seed'.cmk_label.MSK')
               /\ Compound_MAC1' := Hash2(CMK'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1')
               /\ Crypto_bind_request' := Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'
               /\ SND({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)

##### PAC Provisioning #####

8. State = 14
   /\ RCV({Inter_result_tlv.Bind_version.FASTv1.one.CMKnonce.Compound_MAC2'.PAC_type'}_ClientKEY)
   State' := 16 /\ Compound_MAC2' = Hash2(CMK.Bind_version.FASTv1.one.CMKnonce) =|>
               /\ Result_tlv' := new()
               /\ PAC_key' := new()
               /\ PAC_lifetime' := new()
               /\ A_ID' := new()
               /\ PAC_info' := PAC_lifetime'.A_ID'.a_id_info.PAC_type'
               /\ PAC_enc_key' := new()
               /\ PAC_opaque' := {PAC_key'.PAC_info'}_PAC_enc_key'
               /\ SND({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
               /\ secret(PAC_key',sec_packey,{S,P})
               /\ secret(PAC_opaque',sec_pacopaque,{S,P})

9. State = 16 /\ RCV({Result_tlv.PAC_ack'}_ClientKEY) =|>
   State' := 18 /\ SND(eap_success)

end role

#####

role peer(
  S, P : agent,
  Password : symmetric_key,
  Kca : public_key,
  Hash1 : hash_func,

```

```

        Hash2      : hash_func,
        PRF        : hash_func,
        KEYGEN     : hash_func,
        SND, RCV  : channel (dy) )
played_by P def=

local
  State      : nat,
  Snonce    : text,
  Pnonce    : text,
  SessionID : text,
  Csuite    : text,
  PMS       : text,
  CCspec    : text,
  MS        : hash(text.text.text),
  Finished  : hash(hash(text.text.text).agent.agent.text.text.text),
  ClientKEY : hash(agent.text.text.hash(text.text.text)),
  ServerKEY : hash(agent.text.text.hash(text.text.text)),
  Kserver   : public_key,
  UserID    : text,
  FASTv1   : text,
  TLSv1    : text,
  CsuiteList : text,
  SHelloDone : text,
  Schallenge : text,
  Pchallenge : text,
  NTResponse : hash(symmetrical_key.text.text.agent),
  Inter_result_tlv : text,
  Bind_version : text,
  CMKnonce   : text,
  MasterKey  : hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent)),
  MasterSendKey : hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))),
  MasterReceiveKey : hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))),
  MSK        : hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
             hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))),
  Seed       : hash(hash(text.text.text).text.text.text),
  CMK        : hash( hash(hash(text.text.text).text.text.text).text.
                    hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
                    hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))))),
  Compound_MAC1 : hash( hash( hash(hash(text.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
                              .text.text.text.text ) ,
  Compound_MAC2 : hash( hash( hash(hash(text.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))))
                              .text.text.text.text ) ,
  Crypto_bind_request : text.text.text.text.
                    hash( hash( hash(hash(text.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))))
                              .text.text.text.text ) ,
  Crypto_bind_response : text.text.text.text.
                    hash( hash( hash(hash(text.text.text).text.text.text).text.
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))).
                              hash(hash(hash(hash(symmetrical_key)).hash(symmetrical_key.text.text.agent))))
                              .text.text.text.text ) ,
  PAC_type      : text,
  Result_tlv    : text,
  PAC_key       : symmetrical_key,
  PAC_lifetime  : text,
  A_ID          : text,
  PAC_info      : text.text.text.text,
  PAC_opaque    : {symmetrical_key.text.text.text.text}_symmetrical_key,
  PAC_ack       : text

init State := 1

transition

1. State = 1 /\ RCV(eap_request_id) =|>
   State' := 3 /\ UserID' := new()
           /\ SND(UserID')

2. State = 3 /\ RCV(start_fast.FASTv1'.a_id_info) =|>
   State' := 5 /\ TLSv1' := new()
               /\ Pnonce' := new()
               /\ CsuiteList' := new()
               /\ SND(FASTv1'.TLSv1'.session_id_0.Pnonce'.CsuiteList')

3. State = 5 /\ RCV(TLSv1.Snonce'.SessionID'.Csuite'.{S.Kserver'}_inv(Kca).SHelloDone') =|>
   State' := 7 /\ PMS' := new()
               /\ CCspec' := new()
               /\ MS' := PRF(PMS'.Pnonce'.Snonce')

```

```

        /\ Finished' := Hash1(MS'.P.S.Pnonce.Csuite'.SessionID')
        /\ ClientKEY' := KEYGEN(P.Pnonce.Snonce'.MS')
        /\ ServerKEY' := KEYGEN(S.Pnonce.Snonce'.MS')
        /\ SND({PMS'}_Kserver'.CCspec'.{Finished'}_ClientKEY')

##### Phase 2 #####

4. State = 7 /\ RCV(CCspec.{Finished.eap_request_id}_ServerKEY) =|>
   State':= 9 /\ SND({P}_ClientKEY)
              /\ secret(ClientKEY,sec_clientkey,{P,S})
              /\ secret(ServerKEY,sec_serverkey,{P,S})
              /\ request(P,S,nonces,Pnonce.Snonce)

5. State = 9 /\ RCV({Schallenge'}_ServerKEY) =|>
   State':= 11 /\ Pchallenge' := new()
               /\ NTResponse' := Hash2(Password.Pchallenge'.Schallenge'.P)
               /\ SND({Pchallenge'.NTResponse'}_ClientKEY)
               /\ witness(P,S,peer_proof,NTResponse')

6. State = 11 /\ RCV({Hash2(Password.Pchallenge)}_ServerKEY) =|>
   State':= 13 /\ SND({auth_ack}_ClientKEY)
              /\ MasterKey' := Hash2(Hash2(Hash2(Password)).NTResponse)
              /\ MasterSendKey' := PRF(MasterKey')
              /\ MasterReceiveKey' := KEYGEN(MasterKey')
              /\ MSK' := MasterReceiveKey'.MasterSendKey'
              /\ Seed' := PRF(MS.seed_label.Pnonce.Snonce)
              /\ CMK' := PRF(Seed'.cmk_label.MSK')

##### PAC Provisioning #####

7. State = 13
   /\ RCV({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)
   /\ Compound_MAC1' = Hash2(CMK.Bind_version'.FASTv1.zero.CMKnonce') =|>
   State':= 15 /\ Compound_MAC2' := Hash2(CMK.Bind_version'.FASTv1.one.CMKnonce')
              /\ Crypto_bind_response' := Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'
              /\ PAC_type' := new()
   /\ SND({Inter_result_tlv'.Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'.PAC_type'}_ClientKEY)

8. State = 15 /\ RCV({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
   /\ PAC_info' = PAC_lifetime'.A_ID'.a_id_info.PAC_type' =|>
   State':= 17 /\ PAC_ack' := new()
              /\ SND({Result_tlv'.PAC_ack'}_ClientKEY)

9. State = 17 /\ RCV(eap_success) =|>
   State':= 19

end role

#####

role session(
    S, P : agent,
    Password : symmetric_key,
    Kserver : public_key,
    Kca : public_key,
    Hash1 : hash_func,
    Hash2 : hash_func,
    PRF : hash_func,
    KEYGEN : hash_func )
def=
    local PSND,PRCV,SSND,SRCV : channel (dy)

    composition
        server (S,P,Password,Kserver,Kca,Hash1,Hash2,PRF,KEYGEN,SSND,SRCV)
        /\ peer (S,P,Password, Kca,Hash1,Hash2,PRF,KEYGEN,PSND,PRCV)

end role

role environment() def=

const
    eap_request_id, start_fast: text,
    auth_ack, eap_success : text,
    a_id_info : text,
    session_id_0 : text, % SessionID = 0
    seed_label, cmk_label : text,
    zero, one : text,

    sec_packey, sec_pacopaque,
    nonces, peer_proof,
    sec_clientkey,
    sec_serverkey : protocol_id,
    s,p,i : agent,
    kps,kis,kpi : symmetric_key,

```

```

kserver,ki,kca          : public_key,
hash1,hash2            : hash_func,
prf                    : hash_func,
keygen                 : hash_func

intruder_knowledge = { s,p,hash1,hash2,prf,keygen,kca,kserver,ki,inv(ki),kpi,kis }

composition
  session(s,p,kps,kserver,kca,hash1,hash2,prf,keygen)
/\  session(s,i,kis,kserver,kca,hash1,hash2,prf,keygen)
/\  session(i,p,kpi,ki,      kca,hash1,hash2,prf,keygen)

end role

goal

  secrecy_of sec_clientkey, sec_serverkey
  secrecy_of sec_packey, sec_pacopaque

  authentication_on nonces          % server authentication
  authentication_on peer_proof     % peer authentication

end goal

environment()

```

C.1.2 The Output Results

```

root@ebakyt-laptop:/avispa# avispa Server-auth-prov.hlpsl --cl-atse

SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Server-auth-prov.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 632832 states
  Reachable  : 265869 states
  Translation: 0.72 seconds
  Computation: 85.82 seconds

root@ebakyt-laptop:/avispa# avispa Server-auth-prov.hlpsl --typed_model=no --cl-atse

SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  UNTYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Server-auth-prov.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 2238920 states
  Reachable  : 966873 states
  Translation: 0.71 seconds
  Computation: 284.28 seconds

```

C.2 Server-Unauthenticated Provisioning

Phase 1: TLS Tunnel Establishment using Diffie-Hellman Key Exchange.

Phase 2: EAP-FAST-MSCHAPv2 method and Tunnel PAC Provisioning.

C.2.1 The HLPSSL Specification

```

role server(
    S, P      : agent,
    Password  : symmetric_key,
    G         : nat,
    Hash1     : hash_func,
    Hash2     : hash_func,
    PRF       : hash_func,
    KEYGEN    : hash_func,
    PRF1      : hash_func,
    PRF2      : hash_func,
    SND, RCV  : channel (dy) )
played_by S def=

local
    State      : nat,
    Snonce     : text,
    Pnonce     : text,
    SessionID  : text,
    Csuite     : text,
    PMS        : message,
    CCspec     : text,
    MS         : message,
    Finished   : message,
    ClientKEY  : message,
    ServerKEY  : message,
    UserID     : text,
    FASTv1    : text,
    TLSv1     : text,
    CsuiteList : text,
    SHelloDone : text,
    Inter_result_tlv : text,
    Bind_version : text,
    CMKnonce   : text,
    Schallenge : message,
    Pchallenge : message,
    RealServerCH : text,
    RealPeerCH : text,
    NTResponse : message,
    MasterKey   : message,
    MasterSendKey : message,
    MasterReceiveKey : message,
    MSK        : message,
    Seed       : message,
    CMK        : message,
    Compound_MAC1 : message, % server --> peer
    Compound_MAC2 : message, % peer --> server
    Crypto_bind_request : text.text.text.text.message,
    Crypto_bind_response : text.text.text.text.message,
    PAC_type    : text,
    Result_tlv  : text,
    PAC_key     : symmetric_key,
    PAC_lifetime : text,
    A_ID       : text,
    PAC_info    : text.text.text.text,
    PAC_enc_key : symmetric_key,
    PAC_opaque  : {symmetric_key.text.text.text.text}_symmetric_key,
    PAC_ack     : text

init State := 0

transition

1. State = 0 /\ RCV(start) =|>
   State' := 2 /\ SND(eap_request_id)

2. State = 2 /\ RCV(UserID') =|>
   State' := 4 /\ FASTv1' := new()
   /\ SND(start_fast.FASTv1'.a_id_info)

3. State = 4 /\ RCV(FASTv1'.TLSv1'.session_id_0.Pnonce'.CsuiteList') =|>
   State' := 6 /\ Snonce' := new()
   /\ SessionID' := new()
   /\ Csuite' := new()
   /\ SHelloDone' := new()
   /\ SND(TLSv1'.Snonce'.SessionID'.Csuite'.G.exp(G,Snonce').SHelloDone')

```

```

        /\ witness(S,P,nonces,Pnonce'.Snonce')

4. State = 6 /\ RCV(exp(G,Pnonce).CCspec'.{Finished'}_ClientKEY')
/\ Finished' = Hash1(PRF(exp(exp(G,Pnonce),Snonce).Pnonce.Snonce).P.S.Pnonce.Csuite.SessionID)
/\ ClientKEY' = KEYGEN(P.Pnonce.Snonce.PRF(exp(exp(G,Pnonce),Snonce).Pnonce.Snonce)) =|>
State':= 8 /\ PMS' := exp(exp(G,Pnonce),Snonce)
          /\ MS' := PRF(PMS'.Pnonce.Snonce)
          /\ ServerKEY' := KEYGEN(S.Pnonce.Snonce.MS')
          /\ SND(CCspec'.{Finished'.eap_request_id}_ServerKEY')

##### Phase 2 #####

5. State = 8 /\ RCV({P}_ClientKEY) =|>
State':= 10 /\ Schallenge' := PRF1(PMS.Pnonce.Snonce)
          /\ Pchallenge' := PRF2(PMS.Pnonce.Snonce)
          /\ RealServerCH' := new()
          /\ SND({RealServerCH'}_ServerKEY)

6. State = 10 /\ RCV({RealPeerCH'.NTResponse'}_ClientKEY)
          /\ NTResponse' = Hash2(Password.Pchallenge.Schallenge.P) =|>
State':= 12 /\ SND({Hash2(Password.Pchallenge)}_ServerKEY)
          /\ request(S,P,peer_proof,{RealPeerCH'.NTResponse'}_ClientKEY)
          %% /\ request(S,P,peer_proof,NTResponse')

7. State = 12 /\ RCV({auth_ack}_ClientKEY) =|>
State':= 14 /\ Inter_result_tlv' := new() % success
          /\ Bind_version' := new() % same version
          /\ CMKnonce' := new()
          /\ MasterKey' := Hash2(Hash2(Hash2(Password)).NTResponse)
          /\ MasterSendKey' := PRF1(MasterKey')
          /\ MasterReceiveKey' := PRF2(MasterKey')
          /\ MSK' := MasterReceiveKey'.MasterSendKey'
          /\ Seed' := PRF(MS.seed_label.Pnonce.Snonce)
          /\ CMK' := PRF(Seed'.cmk_label.MSK')
          /\ Compound_MAC1' := Hash2(CMK'.Bind_version'.FASTv1.zero.CMKnonce')
          /\ Crypto_bind_request' := Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'
          /\ SND({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)

##### PAC Provisioning #####

8. State = 14
/\ RCV({Inter_result_tlv.Bind_version.FASTv1.one.CMKnonce.Compound_MAC2'.PAC_type'}_ClientKEY)
          /\ Compound_MAC2' = Hash2(CMK.Bind_version.FASTv1.one.CMKnonce) =|>
State':= 16 /\ Result_tlv' := new()
          /\ PAC_key' := new()
          /\ PAC_lifetime' := new()
          /\ A_ID' := new()
          /\ PAC_info' := PAC_lifetime'.A_ID'.a_id_info.PAC_type'
          /\ PAC_enc_key' := new()
          /\ PAC_opaque' := {PAC_key'.PAC_info'}_PAC_enc_key' % PAC-Opaque
          /\ SND({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
          /\ secret(PAC_key',sec_packey,{S,P})
          /\ secret(PAC_opaque',sec_pacopaque,{S,P})

9. State = 16 /\ RCV({Result_tlv.PAC_ack'}_ClientKEY) =|>
State':= 18 /\ SND(eap_success)

end role

#####

role peer(
  S, P : agent,
  Password : symmetric_key,
  G : nat,
  Hash1 : hash_func,
  Hash2 : hash_func,
  PRF : hash_func,
  KEYGEN : hash_func,
  PRF1 : hash_func,
  PRF2 : hash_func,
  SND, RCV : channel (dy) )
played_by P def=

local
  State : nat,
  Snonce : text,
  Pnonce : text,
  SessionID : text,
  Csuite : text,
  PMS : message,
  CCspec : text,
  MS : message,
  Finished : message,

```



```

ClientKEY : message,
ServerKEY : message,
UserID    : text,
FASTv1   : text,
TLSv1    : text,
CsuiteList : text,
SHelloDone : text,
Inter_result_tlv : text,
Bind_version : text,
CMKnonce    : text,
Schallenge  : message,
Pchallenge  : message,
RealServerCH : text,
RealPeerCH  : text,
NTResponse  : message,
MasterKey   : message,
MasterSendKey : message,
MasterReceiveKey : message,
MSK         : message,
Seed        : message,
CMK         : message,
Compound_MAC1 : message, % server --> peer
Compound_MAC2 : message, % peer --> server
Crypto_bind_request : text.text.text.text.message,
Crypto_bind_response : text.text.text.text.message,
PAC_type      : text,
Result_tlv    : text,
PAC_key       : symmetric_key,
PAC_lifetime  : text,
A_ID          : text,
PAC_info      : text.text.text.text,
PAC_opaque    : {symmetric_key.text.text.text.text}_symmetric_key,
PAC_ack       : text

init State := 1

transition

1. State = 1 /\ RCV(eap_request_id) =|>
   State' := 3 /\ UserID' := new()
              /\ SND(UserID')

2. State = 3 /\ RCV(start_fast.FASTv1'.a_id_info) =|>
   State' := 5 /\ TLSv1' := new()
              /\ Pnonce' := new()
              /\ CsuiteList' := new()
              /\ SND(FASTv1'.TLSv1'.session_id_0.Pnonce'.CsuiteList')

3. State = 5 /\ RCV(TLSv1'.Snonce'.SessionID'.Csuite'.G.exp(G,Snonce').SHelloDone') =|>
   State' := 7 /\ PMS' := exp(exp(G,Snonce'),Pnonce)
              /\ CCSpec' := new()
              /\ MS' := PRF(PMS'.Pnonce.Snonce')
              /\ Finished' := Hash1(MS'.P.S.Pnonce.Csuite'.SessionID')
              /\ ClientKEY' := KEYGEN(P.Pnonce.Snonce'.MS')
              /\ ServerKEY' := KEYGEN(S.Pnonce.Snonce'.MS')
              /\ SND(exp(G,Pnonce).CCSpec'.{Finished'}_ClientKEY')

##### Phase 2 #####

4. State = 7 /\ RCV(CCSpec'.{Finished}.eap_request_id)_ServerKEY) =|>
   State' := 9 /\ SND({P}_ClientKEY)
              /\ secret(ClientKEY,sec_clientkey,{P,S})
              /\ secret(ServerKEY,sec_serverkey,{P,S})
              /\ request(P,S,nonces,Pnonce.Snonce)

5. State = 9 /\ RCV({RealServerCH'}_ServerKEY) =|>
   State' := 11 /\ Pchallenge' := PRF2(PMS.Pnonce.Snonce)
              /\ Schallenge' := PRF1(PMS.Pnonce.Snonce)
              /\ RealPeerCH' := new()
              /\ NTResponse' := Hash2(Password.Pchallenge'.Schallenge'.P)
              /\ SND({RealPeerCH'}.NTResponse')_ClientKEY)
              /\ witness(P,S,peer_proof,{RealPeerCH'}.NTResponse')_ClientKEY)
              %% /\ witness(P,S,peer_proof,NTResponse')

6. State = 11 /\ RCV({Hash2(Password.Pchallenge)}_ServerKEY) =|>
   State' := 13 /\ SND({auth_ack}_ClientKEY)
              /\ MasterKey' := Hash2(Hash2(Hash2(Password)).NTResponse)
              /\ MasterSendKey' := PRF1(MasterKey')
              /\ MasterReceiveKey' := PRF2(MasterKey')
              /\ MSK' := MasterReceiveKey'.MasterSendKey'
              /\ Seed' := PRF(MS.seed_label.Pnonce.Snonce)
              /\ CMK' := PRF(Seed'.cmk_label.MSK')

```

```

##### PAC Provisioning #####

7. State = 13
  /\ RCV({Inter_result_tlv'.Bind_version'.FASTv1.zero.CMKnonce'.Compound_MAC1'}_ServerKEY)
     /\ Compound_MAC1' = Hash2(CMK.Bind_version'.FASTv1.zero.CMKnonce') =|>
  State' := 15 /\ Compound_MAC2' := Hash2(CMK.Bind_version'.FASTv1.one.CMKnonce')
              /\ Crypto_bind_response' := Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'
              /\ PAC_type' := new()
  /\ SND({Inter_result_tlv'.Bind_version'.FASTv1.one.CMKnonce'.Compound_MAC2'.PAC_type'}_ClientKEY)

8. State = 15 /\ RCV({Result_tlv'.PAC_key'.PAC_opaque'.PAC_info'}_ServerKEY)
  /\ PAC_info' = PAC_lifetime'.A_ID'.a_id_info.PAC_type' =|>
  State' := 17 /\ PAC_ack' := new()
              /\ SND({Result_tlv'.PAC_ack'}_ClientKEY)

9. State = 17 /\ RCV(eap_success) =|>
  State' := 19

end role

role session(
  S, P      : agent,
  Password  : symmetric_key,
  G         : nat,
  Hash1     : hash_func,
  Hash2     : hash_func,
  PRF       : hash_func,
  KEYGEN    : hash_func,
  PRF1      : hash_func,
  PRF2      : hash_func )
def=
  local PSND, PRCV, SSND, SRCV : channel (dy)

  composition
    server (S, P, Password, G, Hash1, Hash2, PRF, KEYGEN, PRF1, PRF2, SSND, SRCV)
    /\ peer (S, P, Password, G, Hash1, Hash2, PRF, KEYGEN, PRF1, PRF2, PSND, PRCV)

end role

role environment() def=
  const
    eap_request_id, start_fast : text,
    auth_ack, eap_success      : text,
    a_id_info                   : text,
    session_id_0                : text,      % SessionID = 0
    seed_label, cmk_label       : text,
    zero, one                   : text,

    sec_packey, sec_pacopaque,
    nonces, peer_proof,
    sec_clientkey,
    sec_serverkey               : protocol_id,
    s, p, i                     : agent,
    kps, kis, kpi               : symmetric_key,
    g                           : nat,
    hash1, hash2                : hash_func,
    prf                         : hash_func,
    keygen                      : hash_func,
    prf1                        : hash_func,
    prf2                        : hash_func

    intruder_knowledge = { s, p, hash1, hash2, prf, keygen, prf1, prf2, g, kpi, kis }

  composition
    session(s, p, kps, g, hash1, hash2, prf, keygen, prf1, prf2)
    /\ session(s, i, kis, g, hash1, hash2, prf, keygen, prf1, prf2)
    /\ session(i, p, kpi, g, hash1, hash2, prf, keygen, prf1, prf2)

end role

goal
  authentication_on nonces      % server authentication
  authentication_on peer_proof  % peer authentication

  secrecy_of sec_clientkey
  secrecy_of sec_serverkey
  secrecy_of sec_packey
  secrecy_of sec_pacopaque

end goal

environment()

```

C.2.2 The Output Results

When the hlspl tested against the only goal *"authentication_on peer_proof"*, other goals are disabled.

And also when *"witness (P,S,peer_proof,NTResponse)"* and *"request (S,P,peer_proof,NTResponse)"* predicates are used.

```
root@ebakyt-laptop:/avispa# avispa Server-unauth-prov.hlspl --cl-atse
```

SUMMARY

SAFE

DETAILS

BOUNDED_NUMBER_OF_SESSIONS
TYPED_MODEL

PROTOCOL

/avispa/avispa-1.1//testsuite/results/Server-unauth-prov.if

GOAL

As Specified

BACKEND

CL-AtSe

STATISTICS

Analysed : 365 states
Reachable : 263 states
Translation: 0.77 seconds
Computation: 0.00 seconds

When the hlspl tested against the only goal *"authentication_on peer_proof"*, other goals are disabled.

And also when *"witness (P,S,peer_proof,{RealPeerCH'.NTResponse'}_ClientKEY)"* and *"request (S,P,peer_proof,{RealPeerCH'.NTResponse'}_ClientKEY)"* predicates are used.

```
root@ebakyt-laptop:/avispa# avispa Server-unauth-prov.hlspl --cl-atse
```

SUMMARY

UNSAFE

DETAILS

ATTACK_FOUND
TYPED_MODEL

PROTOCOL

/avispa/avispa-1.1//testsuite/results/Server-unauth-prov.if

GOAL

Authentication attack on
(s,p,peer_proof,{RealPeerCH(6)}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2)_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen))

BACKEND

CL-AtSe

STATISTICS

Analysed : 23 states
Reachable : 15 states
Translation: 0.77 seconds
Computation: 0.00 seconds

ATTACK TRACE

```

i -> (s,6): start
(s,6) -> i: eap_request_id

i -> (s,6): UserID(38)
(s,6) -> i: start_fast.n38(FASTv1).a_id_info

i -> (s,3): start
(s,3) -> i: eap_request_id

i -> (s,3): UserID(2)
(s,3) -> i: start_fast.n2(FASTv1).a_id_info

i -> (p,10): eap_request_id
(p,10) -> i: n55(UserID)

i -> (p,10): start_fast.FASTv1(56).a_id_info
(p,10) -> i: FASTv1(56).n56(TLSv1).session_id_0.n56(Pnonce).n56(CsuiteList)

i -> (p,4): eap_request_id
(p,4) -> i: n19(UserID)

i -> (p,4): start_fast.FASTv1(20).a_id_info
(p,4) -> i: FASTv1(20).n20(TLSv1).session_id_0.n20(Pnonce).n20(CsuiteList)

i -> (s,3): n2(FASTv1).TLSv1(3).session_id_0.n20(Pnonce).CsuiteList(3)
(s,3) -> i: TLSv1(3).n3(Snonce).n3(SessionID).n3(Csuite).g.exp(g,n3(Snonce)).
n3(SHelloDone)

i -> (p,4): n20(TLSv1).n3(Snonce).SessionID(21).Csuite(21).g.exp(g,n3(Snonce)).
SHelloDone(21)
(p,4) -> i: exp(g,n20(Pnonce)).n21(CCspec).

{{{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).Csuite(21).SessionID
(21)}_hash1}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_pr
f)_keygen)

i -> (p,4): n21(CCspec).
{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).Csuite(21).SessionID
(21)}_hash1.eap_request_id}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce)
.n3(Snonce)}_prf)_keygen)
(p,4) -> i:

{p}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen
)
    & Add p to set_244; Add s to set_244; Add p to set_245;
    & Add s to set_245;

i -> (p,4):
{RealServerCH(23)}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snocc
e)}_prf)_keygen)
(p,4) -> i:
{n23(RealPeerCH)}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2.{exp(g,n3(Snocc
e)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snocc
e)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen)
&
Witness(p,s,peer_proof,{n23(RealPeerCH)}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snocc
e)}_prf2.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_({p.n20(Pnonce).n3(S
nocc
e)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen));

i -> (s,3): exp(g,n20(Pnonce)).CCspec(4).
{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).n3(Csuite).n3(Sessio
nID)}_hash1}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_pr
f)_keygen)
(s,3) -> i: CCspec(4).
{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).n3(Csuite).n3(Sessio
nID)}_hash1.eap_request_id}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce)
.n3(Snonce)}_prf)_keygen)

i -> (s,3):

{p}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen
)
(s,3) -> i:
{n5(RealServerCH)}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snocc
e)}_prf)_keygen)

i -> (s,3):
{RealPeerCH(6)}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2.{exp(g,n20(Pnocc
e)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnocc
e)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen)
(s,3) -> i:
{{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}_hash2}_({s.n20(Pnonce).n3(Snocc
e)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf)_keygen)

```

```

&
Request(s,p,peer_proof,{RealPeerCH(6)}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}
_prf2.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_{p.n20(Pnonce).n3(Sn
once).{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen));

```

When the hlp1 tested against the only goal "**secrecy_of sec_packey**", other goals are disabled.

```
root@ebakyt-laptop:/avispa# avispa Server-unauth-prov.hlp1 --cl-atse
```

```
SUMMARY
UNSAFE
```

```
DETAILS
ATTACK FOUND
TYPED_MODEL
```

```
PROTOCOL
/avispa/avispa-1.1//testsuite/results/Server-unauth-prov.if
```

```
GOAL
Secrecy attack on (n8(PAC_key))
```

```
BACKEND
CL-AtSe
```

```
STATISTICS
```

```

Analyzed : 108 states
Reachable : 41 states
Translation: 0.77 seconds
Computation: 0.00 seconds

```

```
ATTACK TRACE
```

```

i -> (s,3): start
(s,3) -> i: eap_request_id

i -> (s,3): UserID(2)
(s,3) -> i: start_fast.n2(FASTv1).a_id_info

i -> (p,4): eap_request_id
(p,4) -> i: n19(UserID)

i -> (p,4): start_fast.n2(FASTv1).a_id_info
(p,4) -> i: n2(FASTv1).n20(TLSv1).session_id_0.n20(Pnonce).n20(CsuiteList)

i -> (s,3): n2(FASTv1).TLSv1(3).session_id_0.n20(Pnonce).CsuiteList(3)
(s,3) -> i: TLSv1(3).n3(Snonce).n3(SessionID).n3(Csuite).g.exp(g,n3(Snonce)).
n3(SHelloDone)

i -> (p,4): n20(TLSv1).n3(Snonce).SessionID(21).Csuite(21).g.exp(g,n3(Snonce)).
SHelloDone(21)
(p,4) -> i: exp(g,n20(Pnonce)).n21(CCspec).

{{{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).Csuite(21).SessionID
(21)}_hash1}_{p.n20(Pnonce).n3(Snonce).{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_pr
f}_keygen)

i -> (p,4): n21(CCspec).

{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).Csuite(21).SessionID
(21)}_hash1.eap_request_id}_{s.n20(Pnonce).n3(Snonce).{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce)
.n3(Snonce)}_prf}_keygen)
(p,4) -> i:
{p}_{p.n20(Pnonce).n3(Snonce).{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen
)
& Add p to set_240; Add s to set_240; Add p to set_241;
& Add s to set_241;

i -> (p,4):
{RealServerCH(23)}_{s.n20(Pnonce).n3(Snonce).{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonc
e)}_prf}_keygen)
(p,4) -> i:
{n23(RealPeerCH)}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n3(Snonc

```

```

e)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (s,3): exp(g,n20(Pnonce)).CCspec(4).

{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).n3(Csuite).n3(SessionID)}_hash1}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(s,3) -> i: CCspec(4).

{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.p.s.n20(Pnonce).n3(Csuite).n3(SessionID)}_hash1.eap_request_id}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (s,3):
{p}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(s,3) -> i:
{n5(RealServerCH)}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (s,3):
{RealPeerCH(6)}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(s,3) -> i:
{{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}_hash2}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (p,4):
{{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2}_hash2}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(p,4) -> i:
{auth_ack}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (s,3):
{auth_ack}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(s,3) -> i:
{n7(Inter_result_tlv).n7(Bind_version).n2(FASTv1).zero.n7(CMKnonce).{{{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf.seed_label.n20(Pnonce).n3(Snonce)}_prf.cmk_label.{{{kps}_hash2}_hash2}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf2.{{{kps}_hash2}_hash2}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf1}_prf.n7(Bind_version).n2(FASTv1).zero.n7(CMKnonce)}_hash2}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (p,4):
{Inter_result_tlv(25).n7(Bind_version).n2(FASTv1).zero.n7(CMKnonce).{{{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf.seed_label.n20(Pnonce).n3(Snonce)}_prf.cmk_label.{{{kps}_hash2}_hash2}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf2.{{{kps}_hash2}_hash2}.{kps.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf1}_prf.n7(Bind_version).n2(FASTv1).zero.n7(CMKnonce)}_hash2}.n25(PAC_type)}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n3(Snonce)*n20(Pnonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)

i -> (s,3):
{n7(Inter_result_tlv).n7(Bind_version).n2(FASTv1).one.n7(CMKnonce).{{{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf.seed_label.n20(Pnonce).n3(Snonce)}_prf.cmk_label.{{{kps}_hash2}_hash2}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf2.{{{kps}_hash2}_hash2}.{kps.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf2}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf1.p}_hash2}_hash2}_prf1}_prf.n7(Bind_version).n2(FASTv1).one.n7(CMKnonce)}_hash2}.PAC_type(8)}_({p.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
(s,3) -> i:
{n8(Result_tlv).n8(PAC_key)}.{n8(PAC_key).n8(PAC_lifetime).n8(A_ID).a_id_info.PAC_type(8)}_n8(PAC_enc_key).n8(PAC_lifetime).n8(A_ID).a_id_info.PAC_type(8)}_({s.n20(Pnonce).n3(Snonce)}.{exp(g,n20(Pnonce)*n3(Snonce)).n20(Pnonce).n3(Snonce)}_prf}_keygen)
& Secret(n8(PAC_key),set_189); Add s to set_189;
& Add p to set_189; Add s to set_190; Add p to set_190;

```

APPENDIX D

The Four-Way Handshake Protocol

D.1 The HLPSL Specification

```

role alice(
  A, B      : agent,
  PMK       : symmetric_key,
  PTK_PRF   : hash_func,
  MIC_Hash  : hash_func,          % MAC hash func
  Succ      : hash_func,
  H_MAC     : hash_func,
  Snd, Rcv  : channel(dy)      )
played_by A def=

local
  State      : nat,
  Anonce     : text,
  Snonce     : text,
  Sqn        : text,
  GMK        : text,
  GNonce     : text,
  GTK_PRF    : hash_func,
  GTK        : hash(text.agent.text),
  A_rsnie    : text,
  B_rsnie    : text,
  PMKID      : hash(symmetric_key.agent.agent),
  PTK        : hash(symmetric_key.agent.agent.text.text),
  MIC1       : hash(symmetric_key.text.text.text),
  % MIC2      : hash( hash(symmetric_key.agent.agent.text.text).text.hash(text).text.
  %             {hash(text.agent.text)}_hash(symmetric_key.agent.agent.text.text) ),
  MIC2       : message,
  MIC3       : hash(hash(symmetric_key.agent.agent.text.text).hash(text))

init
  State := 0

transition

1. State = 0 /\ Rcv(start) =|>
   State' := 2 /\ Anonce' := new()
              /\ PMKID'  := H_MAC(PMK.A.B)
              /\ Sqn'    := new()
              /\ Snd(Anonce'.Sqn'.PMKID')
              /\ witness(A,B,bob_alice_na,Anonce')

2. State = 2 /\ Rcv(Snonce'.B_rsnie'.Sqn.MIC1')
              /\ MIC1' = MIC_Hash(PMK.Snonce'.Sqn.B_rsnie') =|>
   State' := 4 /\ A_rsnie' := new()
              /\ GMK'     := new()
              /\ GNonce'  := new()
              /\ GTK'     := GTK_PRF(GMK'.A.GNonce')
              /\ PTK'     := PTK_PRF(PMK.A.B.Anonce.Snonce')
              /\ MIC2'    := MIC_Hash(PTK'.Anonce.Succ(Sqn).A_rsnie'.{GTK'}_PTK')
              /\ Snd(Anonce.A_rsnie'.{GTK'}_PTK'.Succ(Sqn).MIC2')
              /\ secret(GTK',gtk1,{A,B})

3. State = 4 /\ Rcv(Succ(Sqn).MIC3')
              /\ MIC3' = MIC_Hash(PTK.Succ(Sqn)) =|>
   State' := 6 /\ request(A,B,alice_bob_ns,Snonce)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role bob(
  A, B      : agent,
  PMK       : symmetric_key,
  PTK_PRF   : hash_func,
  MIC_Hash  : hash_func,
  Succ      : hash_func,
  H_MAC     : hash_func,
  Snd, Rcv  : channel(dy)      )
played_by B def=

local
  State      : nat,
  Anonce     : text,
  Snonce     : text,

```

```

Sqn      : text,
GTK      : hash(text.agent.text),
A_rsnie  : text,
B_rsnie  : text,
PMKID    : hash(symmetric_key.agent.agent),
PTK      : hash(symmetric_key.agent.agent.text.text),
MIC1     : hash(symmetric_key.text.text.text),
% MIC2    : hash( hash(symmetric_key.agent.agent.text.text).text.hash(text).text.
%           {hash(text.agent.text)}_hash(symmetric_key.agent.agent.text.text) ),
MIC2     : message,
MIC3     : hash(hash(symmetric_key.agent.agent.text.text).hash(text))

init
  State := 1

transition

1. State = 1 /\ Rcv(Anonce'.Sqn'.PMKID') =|>
   State':= 3 /\ Snonce' := new()
                /\ B_rsnie' := new()
                /\ PTK'      := PTK_PRF(PMK.A.B.Anonce'.Snonce')
                /\ MIC1'     := MIC_Hash(PMK.Snonce'.Sqn'.B_rsnie')
                /\ Snd(Snonce'.B_rsnie'.Sqn'.MIC1')
                /\ witness(B,A,alice_bob_ns,Snonce')

2. State = 3 /\ Rcv(Anonce.A_rsnie'.{GTK'}_PTK'.Succ(Sqn).MIC2')
               /\ MIC2' = MIC_Hash(PTK.Anonce.Succ(Sqn).A_rsnie'.{GTK'}_PTK') =|>
   State':= 5 /\ MIC3' := MIC_Hash(PTK.Succ(Sqn))
               /\ Snd(Succ(Sqn).MIC3')
               /\ request(B,A,bob_alice_na,Anonce)

end role

role session(
  A, B      : agent,
  PMK      : symmetric_key,
  PTK_PRF  : hash_func,
  MIC_Hash : hash_func,
  Succ     : hash_func,
  H_MAC    : hash_func )
def=

local
  SA, RA, SB, RB : channel (dy)

composition
  alice (A,B,PMK, PTK_PRF, MIC_Hash, Succ, H_MAC, SA,RA)
  /\ bob  (A,B,PMK, PTK_PRF, MIC_Hash, Succ, H_MAC, SB,RB)

end role

role environment()
def=

const
  a, b      : agent,
  gtk1,
  alice_bob_ns,
  bob_alice_na : protocol_id,
  pmk_a_b,
  pmk_a_i,
  pmk_i_b      : symmetric_key,
  ptk_prf      : hash_func,
  mic_hash     : hash_func,
  succ         : hash_func,
  h_mac        : hash_func

intruder_knowledge = {a,b,ptk_prf,mic_hash,succ,h_mac,pmk_a_i,pmk_i_b}

composition
  session(a,b, pmk_a_b, ptk_prf, mic_hash, succ, h_mac)
  /\ session(a,i, pmk_a_i, ptk_prf, mic_hash, succ, h_mac)
  /\ session(i,b, pmk_i_b, ptk_prf, mic_hash, succ, h_mac)

end role

goal
  secrecy_of gtk1
  authentication_on alice_bob_ns
  authentication_on bob_alice_na

end goal

environment()

```


D.2 The Output Results

```
root@ebakyt-laptop:/avispa# avispa Four-way-handshake.hlpsl --cl-atse
```

```
SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Four-way-handshake.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 13 states
  Reachable  : 7 states
  Translation: 0.02 seconds
  Computation: 0.00 seconds
```

```
root@ebakyt-laptop:/avispa# avispa Four-way-handshake.hlpsl --typed_model=no --cl-atse
```

```
SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  UNTYPED_MODEL

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Four-way-handshake.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 56 states
  Reachable  : 33 states
  Translation: 0.01 seconds
  Computation: 0.00 seconds
```

```
root@ebakyt-laptop:/avispa# avispa Four-way-handshake.hlpsl --ofmc
```

```
% OFMC
% Version of 2006/02/13

SUMMARY
SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS

PROTOCOL
  /avispa/avispa-1.1//testsuite/results/Four-way-handshake.if

GOAL
  as_specified

BACKEND
  OFMC

COMMENTS
```

STATISTICS

```
parseTime: 0.00s
searchTime: 5.13s
visitedNodes: 1588 nodes
depth: 10 plies
```

```
root@ebakyt-laptop:/avispa# avispa Four-way-handshake.hlp1 --typed_model=no --ofmc
```

```
% OFMC
```

```
% Version of 2006/02/13
```

```
SUMMARY
```

```
SAFE
```

```
DETAILS
```

```
BOUNDED_NUMBER_OF_SESSIONS
```

```
PROTOCOL
```

```
/avispa/avispa-1.1//testsuite/results/Four-way-handshake.if
```

```
GOAL
```

```
as_specified
```

```
BACKEND
```

```
OFMC
```

```
COMMENTS
```

```
STATISTICS
```

```
parseTime: 0.00s
searchTime: 5.32s
visitedNodes: 2220 nodes
depth: 10 plies
```

