國 立 交 通 大 學

電機與控制工程學系

碩士論文

設計與實作基於單次密鑰加密之無線網路認證協定

Design and Implementation of Secure Wireless Authentication Protocol using One-Time Key

研 究 生：陸培華

Student: Pei-Hua Lu

指導教授：黃育綸　博士

Advisor: Dr. Yu-Lun Huang

中華民國九十七年八月

August, 2008

設計與實作基於單次密鑰加密之無線網路認證協定

Design and Implementation of Secure Wireless Authentication Protocol using One-Time Key

研 究 生：陸培華　　Student: Pei-Hua Lu

指導教授：黃育綸　博士　　Advisor: Dr. Yu-Lun Huang

國 立 交 通 大 學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in partial Fulfill of the Requirements

for the Degree of

Master

in

Department of Electrical and Control Engineering

August, 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

設計與實作基於單次密鑰加密之無線網路認證協定

學生：陸培華　　指導教授：黃育綸　博士

國 立 交 通 大 學電機與控制工程學系（研究所）碩士班

摘　　要

換手的安全和效率問題變得越來越具重要性在現代的無線網路環境中。 在安全及效率中取得平衡是需要被考量的。 我們提出了一個新的協定，使用單次密鑰來做為使用者認證。 這個提出的新協定可以有效的支援同領域及跨領域認證。 我們利用金鑰發行中心(KDC)來管理使用者和授權伺服器。 此協定需要五個訊息來達到同領域的初始認證； 三個訊息完成後繼認證； 以及五個給換手認證。 在換手的過程中不需要金鑰發行中心可減輕金鑰發行中心的負擔。 我們實現一個整合802.1X和此協議的擴展認證協議(EAP)，並和其他擴展認證協議做比較。 這結果也給了一個應用我們的協議到現存的802.11無線網路的簡單方法。 最後，此協定被BAN邏輯所證明其正確性。

Design and Implementation of Secure Wireless Authentication Protocol using One-Time Key

Student: Pei-Hua Lu Advisor: Dr. Yu-Lun Huang

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

Handover security and efficiency have become more and more important in modern wireless network designs. Balance between security and efficiency needs to be considered. We propose a new protocol using one-time keys for user authentication, called OSNP. The proposed protocol can support both intra-domain and inter-domain authentications efficiently. Our protocol uses KDC (Key Distributed Center) to manage both users and authorization servers. It requires five messages for intra-domain initial authentication; three for subsequent authentication; and five for handover authentication. No KDC is needed during a handover and our design reduces the load on it. We show an integration of EAP method from 802.1X and our protocol, comparing the result with other EAP methods. It also gives an easy way to apply our protocol in existing 802.11 wireless networks. In final, the protocol logic is proved by BAN logic and its enhancement.

誌謝

在這裡首先要感謝我的指導老師，黃育綸博士認真和不怕麻煩的教導我這個庸才。 除了研究之外，對於寫作完全不行的我也能一步一步的從頭教起。 也很感謝iCAST (The International Collaboration for Advancing Security Technology)計劃能夠提供一個很棒的機會，讓我能夠赴美國加州柏克萊分校 (University of California, Berkeley)，與當地的教授們，Doug Tygar和Anthony D. Joseph進行這一年來的很棒的合作研究。 而在柏克萊同計劃的同學彼此的互相勉勵更是完成此研究的一大推力。 時常的喝茶聊天更是抒解研究壓力的好活動。 並感謝黃詠文學長在技術上的大力協助，初步的實驗環境和系統建立上沒有他是不行的。 也很感謝幫忙建立SWOON平台的新竹交大研發團隊，能夠做出如此出色的平台。 再最後也感謝RTES Lab的全體。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We propose a handover authentication protocol for WiFi (802.11) networks. Our protocol does not require a public key infrastructure and can be integrated with the 802.1X [1] Extensible Authentication Protocol (EAP) for WLANs[2] [3].

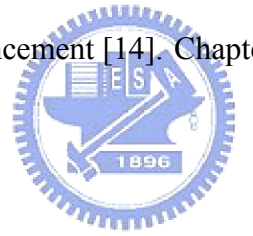
To enhance the security in wireless networks, 802.11i [4] defined a new security model for 802.11 a/b/g networks, specified new standards for authentication, encryption and message integrity, and implemented 802.1X [1] for user authentication and key distribution. 802.1X is a port-based network access control mechanism that provides EAP and can be used in conjunction with other mature authentication protocols, such as TLS, PEAP, CHAP, etc.

There are many EAP methods supporting 802.11i authentication, including EAP-TLS [5] [6], EAP-FAST [7], and LEAP. EAP-TLS and EAP-FAST are used with public-key cryptography for authentication. Compared to symmetric-key systems, public-key systems and certificates produce stronger security, but require more computational power. LEAP, a symmetric-key authentication protocol, requires less computational power and thus takes less response time when performing user authentication. However, LEAP is vulnerable to several attacks [8] such as weak encryption keys. To balance efficiency and security, an efficient authentication is required for wireless networks, especially for roaming users.

There are many methods to approach the balance between efficiency and security by using Kerberos [9]. In 2007, Zrelli et al. [10] presented an integration of the Kerberos protocol with EAP framework, called EAP-Kerberos. Kerberos is known for its symmetric-key

cryptography, strong per-person key and inter-domain authentication. However, it is still inefficient in the WLAN environment bacause users need to use proxies to get tickets from the Kerberos Key Distributed Center (KDC). In other words, in Kerberos, KDC is involved in the handover of a roaming user. And Kerberos is also vulnerable to weak password-based attacks, discussed in [11] [12].

We propose a more efficient authentication protocol supporting handover authentication without a trusted of third-party. The paper is organized as follows. Related research is detailed in Chapter 2. We present our authentication protocol and the integration with EAP in Chapter 3 and 4, respectively. Chapter 5 demonstrates intra-domain authentication methods of EAP-OSNP by a experiment, comparing it with EAP methods. Chapter 6 analyses our system to others and formally proves that our protocol can reach the goals of mutual authentication by using BAN logic [13] and its enhancement [14]. Chapter 7 concludes the paper.



Chapter 2

Related Work

2.1 Network Authentication Protocols

This section summarizes the characteristics and drawbacks of some related authentication protocols.

- Kerberos

  Kerberos [9] was developed as a solution to network security problems, such as replaying, eavesdropping and sniffing packets. In Kerberos V5, seven messages are required for initial intra-domain authentication. The number of message required for inter-domain authentication depends on the number of KDCs between the visited and home domains.

- One-Time Password/Kerberos

  Since the traditional password authentication is vulnerable to dictionary and playback attacks, in 2005, Cheng et al. [15] presented a new authentication method that integrates the Kerberos protocol and a one-time password (OTP) system. The main idea of OTP authentication is to add random factors during the initial login process and make the password used vary from time to time. Similar to the Kerberos protocol, the OTP/ Kerberos protocol requires three steps to authenticate a user: authentication by the KDC, request of tickets from the Ticket-Granting Server (TGS) and access to the server (S).

On the client, the OTP is generated by hashing the user's secret passphrase and the seed from the KDC. By encrypting and decrypting messages with the OTP, the user and server mutually authenticate each other. However, to generate an OTP for authentication requires seven messages in the first step mentioned above. In other words, OTP/ Kerberos increases the communication cost for authenticating a user, resulting in longer user authentication time, which is not practical for roaming users in wireless networks.

- Secure Network Protocol

  In 1999, Shieh et al. [14] proposed a symmetric-key based protocol, Secure Network Protocol (SNP), to provide an efficient way for both intra- and inter- domain authentication. Compared to Kerberos, fewer messages are required in SNP to authenticate client identity. For intra-domain authentication, SNP takes four messages to authentication client identity and one more optional message for mutually authenticating the server. For inter-domain authentication, it takes seven messages for initial authentication, regardless of the number of hops between the visited and home domains. Only two messages are required for subsequent authentication when requesting the same service. To simplify the design, SNP replaces timestamps with nonces, reducing the need for time servers. For faster authentication, a master key is shared by the authentication server (AS) and the service servers (S). The unchanged master keys can make the system vulnerable to various attacks.

2.2 EAP-based Authentication Protocols

EAP is an authentication framework used in various networks, such as WLANs and Point-to-Point connections (PPP). EAP provides some common functions and a negotiation of

the desired authentication methods, such as EAP-MD5, EAP-OTP, EAP-TLS, etc. In WLANs, EAP authentication methods are normally supported with Remote Authentication Dial-In User Service (RADIUS) [16] [17] [18] [19]. RADIUS is also a client/server protocol that enables remote access servers to communicate with a centralized authentication server to authenticate dial-in users. It also authorizes their access to the requested services. The RADIUS server supporting various EAP methods then becomes the major authority of wireless networks. This section summarizes EAP authentication methods supporting strong authentication for roaming users in WLANs.

- EAP-TLS

  EAP-TLS [5] [6] is a popular EAP method for securing WLANs with RADIUS. The mobile node and RADIUS server must have certificates to mutually authenticate each other. EAP-TLS is resilient to man-in-the-middle attacks. However, it requires a trusted-third party (Certificate Authority) to support authentication between the mobile node and RADIUS server. Also, it requires extra management for administrating and distributing certificates, supported by cooperative network management systems (NMS) or Operation, Administration, Maintenance and Provisioning (OAM&P).

- EAP-Kerberos

  In 2007, S. Zrelli and Y. Shinoda [10] showed how to integrate the Kerberos protocol as an authentication method in EAP-based authentication frameworks. They define the architectural elements and specify the encapsulation of the Kerberos messages in EAP packets. Such a design allows a mobile node to be authenticated using the Kerberos systems. When a mobile node, for example, issues an initial authentication request, the EAP-Kerberos client encapsulates the Kerberos messages into EAP packets and sends them to the access node. The access node then delivers these EAP packets to the

RADIUS server via the AAA (Authentication Authorization and Accounting) protocol. The server either validates these messages or forwards them to the Kerberos KDC, as shown in Fig. 2.1.

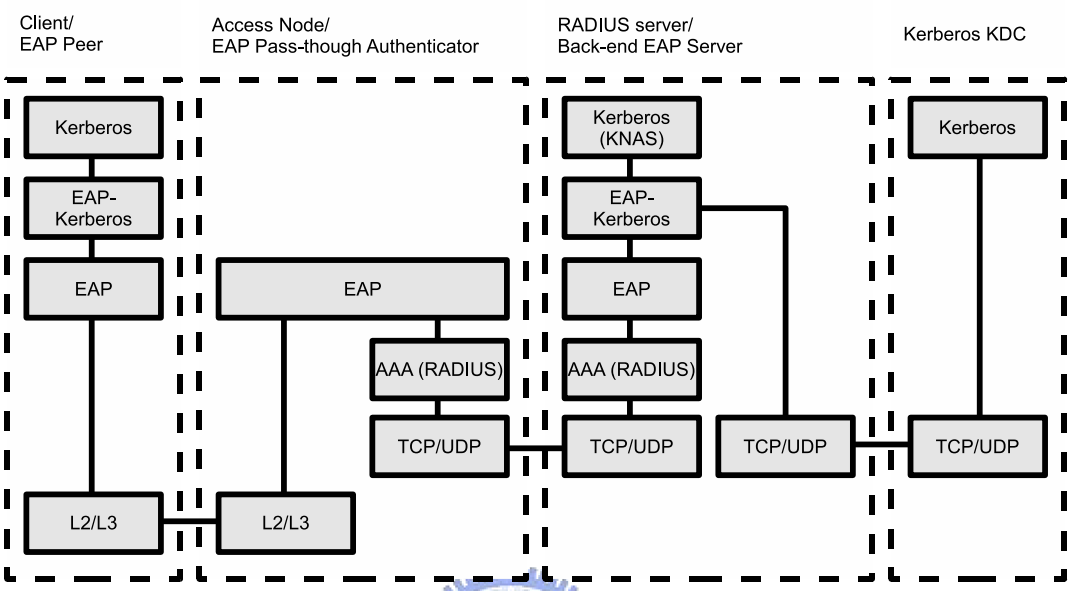


Figure 2.1: Protocol stacks of the components of the EAP-Kerberos authentication framework.

- Kerberized Handover Keying (KHK)

In 2007, Ohba et al. [20] proposed a Kerberized media-independent handover key management architecture for the existing link-layer technologies, including 802.11 and 802.16. The architecture uses Kerberos for securing key distribution between a mobile node, an access point (authenticator) and a server. In KHK, two handover modes are presented: proactive and reactive. In proactive mode, a mobile node uses a pre-obtained credential to authenticate with the access point. In reactive mode, the access point acts as a Kerberos client on behalf of the mobile node. In this architecture, Kerberos can be bootstrapped from initial authentication using an EAP method. This makes KHK work across multiple AAA domains. However, similar to Kerberos, the KDC is involved in handover authentication in KHK. Thus, the handover performance for reactive mode depends on the location of KDC. The larger the distance between a KDC and a mobile

node, the longer the time required for a handover authentication. Also, such an architecture incurs extra costs for setting up time servers for synchronizing machine time in the network, as mentioned in the previous chapter.

Chapter 3

Proposed Protocol: OSNP

We propose the integration of SNP, OTP and EAP for authenticating IEEE 802.11 mobile nodes, giving us support for fast roaming —— One-time key Secure Network Protocol (OSNP).

3.1 Preliminaries

Table 3.1 shows the abbreviations and symbols used in our protocol. Similar to other password-based authentication methods, our authentication servers or KDCs share secrets with users and servers in their own domains. For example, the user chooses his own strong password and shares it with his KDC; the server chooses its own strong password and shares it with its KDC. These shared secrets are assumed to be stored in a secure storage system.

3.2 Intra-Domain Authentication

In our protocol, we give three methods for three types of intra-domain authentication: initial, subsequent and handover authentication. In initial authentication, five messages are required for mutually authenticating the user and server. To subsequently authenticate with the same server, only three messages are used (we renew session keys without querying the KDC.) Handover authentication requires five messages to renegotiate a new session key with another server of the domain. Although five messages are required, the KDC is not involved, reducing its load.

Since authentication occurs in a common domain, the notations U_a, S_a, and KDC_a are

Table 3.1: Message Abbreviations

Abbreviation	Description
TKT_X	Ticket issued by X
CH_X	Challenge issued by X
$RESP_X$	Response to CH_X
A_X	Authenticator issued by X
$authRQ_X$	Authentication request sent by entity X
$authAK_X$	Authentication response to $authRQ_X$
$sauthRQ_X$	Subsequent authentication request issued by X
$sauthAK_X$	Authentication response to $sauthRQ_X$
$hauthRQ_X$	Handover authentication request issued by X
$hauthAK_X$	Authentication response to $hauthRQ_X$
$hauthVF_X$	The verifier sent by the previous server X
$iauthRQ_X$	Inter-domain authentication request issued by X
$iauthAK_X$	Authentication response to $iauthRQ_X$
$iauthFW_X$	Inter-domain authentication forwarding to X
U_a	User principal in domain "a"
S_a	Server principal in domain "a"
KDC_a	Key distribution center in domain "a"
PW_X	Password of entity X
N_X	Nonce generated by X
K_{SS}	Session key to secure the communication
K_g	Group key for all servers under a KDC
K_{TU}	Temporary user key for subsequent authentication
OTK_X	One-time key of entity X
SID	Session identification
VT_X	Local time of entity X
TU_X	Temporary user identity of entity X
rt	The remaining time of the validate ticket
ct	The current time of the local host

simplified to U, S, and KDC, respectively.

3.2.1 Initial Authentication

To initially access a server, the wireless client U sends the authentication request message (M1) to the server. The message is then forwarded to the KDC with server credentials (M2). The KDC authenticates the identities of user and server; generates a session key and sends the message (M3) back to the server. The server then forwards the message with encrypted session key (M4) to the user. The final acknowledge (M5) is sent back to the server for mutual authentication. Fig. 3.1 shows the message flow of intra-domain initial authentication.

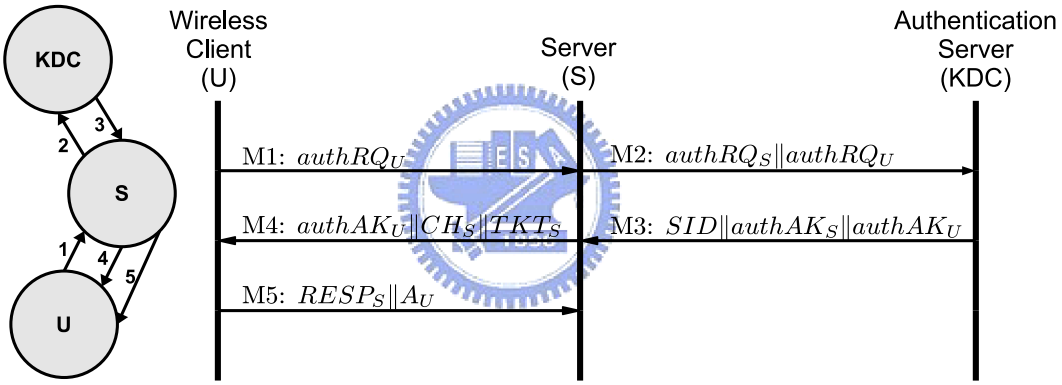


Figure 3.1: The intra-domain initial authentication protocol.

M1. $U \rightarrow S : authRQ_U$

When the authentication process starts, the user generates an authentication request, containing a nonce N_U, the user's identity and an encrypted message using user's one-time key OTK_U: $authRQ_U = U||N_U||\{U, N_U\}_{OTK_U}$. The one-time key is a hashed value of user's identity, password and nonce.

$$OTK_U = Hash(U, N_U, PW_U)$$

The request is sent to the server for authentication.

M2. $S \rightarrow KDC : authRQ_S \| authRQ_U$

After receiving the user request, S generates its request $authRQ_S = S \| N_S \| \{S, N_S\}_{OTK_S}$. It then concatenates the two requests and sends them to KDC.

M3. $KDC \rightarrow S : SID \| authAK_S \| authAK_U$

To identify each session, the KDC generates a unique identity $SID = U \| S \| N_U$ for each session after receiving the authentication requests. It also calculates the one-time keys OTK_U and OTK_S to verify the requests. After authenticating both identities, KDC randomly generates a session key K_{SS}. The session key, server's nonce, and identity are encrypted with OTK_S to acknowledge the server's authentication request.

$$authAK_S = \{N_S, U, K_{SS}\}_{OTK_S}$$

A temporary user key K_{TU} is generated and encrypted with OTK_U in the acknowledgement.

$$authAK_U = \{N_U, S, K_{SS}, K_{TU}\}_{OTK_U}$$

The temporary user key can be used for subsequent authentication.

M4. $S \rightarrow U : authAK_U \| CH_S \| TKT_S$

Upon receiving the response from KDC, S decrypts $authAK_S$ in M3 with OTK_S and gets the session key. The server generates a new challenge for authenticating the user. The new challenge is made by encrypting a new nonce N'_S and the server's identity with the session key K_{SS}, represented as $CH_S = \{S, N'_S\}_{K_{SS}}$. In addition, S can also optionally generate a ticket for subsequently authenticating the same user.

$$TKT_S = SID \| \{U, VT_S, K_{SS}\}_{OTK_S}$$

VT_S is a validation time for TKT_S. Since the validation of a ticket is determined by its issuer, no time server is required.

M5. $U \rightarrow S : RESP_S \| A_U$

The user receives the session key after decrypting $authAK_U$. It then generates a response $RESP_S = \{U, N'_S\}_{K_{SS}}$ to CH_S. The response is generated by replacing the server's identity with the user's. Then, the mutual authentication of the user and server can be guaranteed by encrypting and decrypting these messages with the shared session key. In addition, a temporary authenticator $A_U = \{S, VT_U, K_{SS}\}_{K_{TU}}$ is also appended to the response message. The authenticator can be used to authenticate the user in subsequent authentication rounds without querying KDC. As above, no time server is needed.

3.2.2 Subsequent Authentication

Subsequent authentication rounds occur when a user requests the same services within the specified time. For intra-domain subsequent authentication, the user must send the ticket and his temporary credential to the server, as shown in Fig. 3.2. Below is the flow for intra-domain subsequent authentication.

M1. $U \rightarrow S : sauthRQ_U$

To initiate a subsequent authentication, the user generates a subsequent authentication request, consisting of a nonce and a ticket, and sends it to the server. The subsequent authentication request can be represented as $sauthRQ_U = N_U \| TKT_S$.

M2. $S \rightarrow U : sauthAK_U \| CH_S \| A_U$

After receiving the $sauthRQ_U$, the server retrieves the user identity from the SID

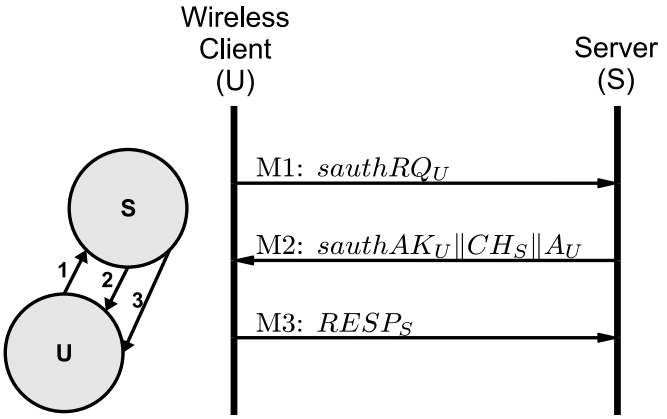


Figure 3.2: The intra-domain subsequent authentication protocol.

contained in TKT_S. Then, the server decrypts the ticket and checks its validation time VT_S. If the ticket is not expired, the server generates a nonce and a new session key. The user nonce and a new session key are then encrypted with the previous session key to acknowledge the request from user.

$$sauthAK_U = \{N_U, K'_{SS}\}_{K_{SS}}$$

Then, the server nonce and identity are encrypted with the new session key. This is a new challenge for mutually authentication the user.

$$CH_S = \{S, N_S\}_{K'_{SS}}$$

A concatenation of $sauthAK_U$, CH_S with the temporary authenticator A_U received in the initial authentication is then sent back to the user.

M3. $U \rightarrow S : RESP_S$

The user decrypts the temporary authenticator A_U to get VT_U and K_{SS}. It checks the validation time of the temporary authenticator, if the authenticator is not expired, the user decrypts $sauthAK_S$, and obtains the new session key. The nonce N_S and the user identity are then encrypted using the new session key to respond the CH_S. The subsequent response is represented as $RESP_S = \{U, N_S\}_{K'_{SS}}$.

3.2.3 Handover Authentication

Handover authentication occurs when a user requests a server belonging to the same domain as the previous server. In most network authentication protocols, an initial authentication is required when contacting another server. This increases the load on the KDC. In such a case, since the user is already authenticated by the KDC and recognized by the previous server, re-authentication can be performed by the previous server to reduce the load on the KDC. In this paper, we propose a 5-step handover authentication protocol for intra-domain authentication. Fig. 3.3 illustrates the message flow.

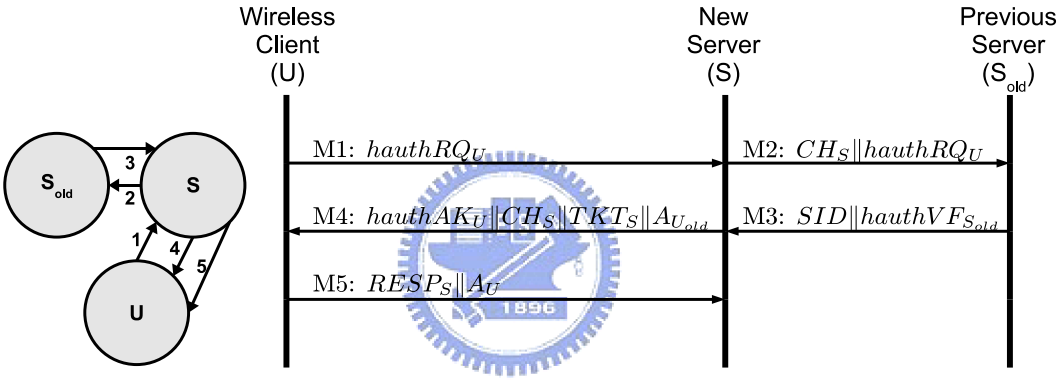


Figure 3.3: The intra-domain handover authentication protocol.

M1. $U \rightarrow S : hauthRQ_U$

Similar to subsequent authentication, a user sends a $hauthRQ_U$ to initiate a handover authentication. The $hauthRQ_U$ is the same as $sauthRQ_U$, containing a user identity, nonce and ticket to the previous server.

$$hauthRQ_U = U||N_U||TKT_{S_{old}}$$

M2. $S \rightarrow S_{old} : CH_S||hauthRQ_U$

S generates a $CH_S = \{S, N_S\}_{K_g}$ and forwards it together with the $hauthRQ_U$ to its previous server S_{old} in $TKT_{S_{old}}$.

M3. $S_{old} \rightarrow S : SID \| hoauthVF_{S_{old}}$

After validating the ticket, S_{old} retrieves the user identity, validation time $VT_{S_{old}}$ and the previous session key $K_{SS_{old}}$ from the $TKT_{S_{old}}$. The server S_{old} then calculates the remaining validation time for the ticket:

$$rt = ct - VT_{S_{old}}$$

The remaining validation time, user identity and the previous session key are encrypted together with the response to CH_S and the temporary authenticator $A_{U_{old}}$ using the group key K_g. $hauthVF_{S_{old}} = \{N_S, U, K_{SS_{old}}, rt, A_{U_{old}}\}_{K_g}$ is sent to S securely.

M4. $S \rightarrow U : hauthAK_U \| CH'_S \| TKT_S \| A_{U_{old}}$

Upon receiving M3, S decrypts the message with the group key and gets the previous session key, the temporary authenticator and the remaining validation time of the previous ticket. The temporary authenticator will be forwarded to the user for proving the user's identity. When generating the new ticket TKT_S for the user, S calculates its validation time according to the remaining validation time.

$$VT_S = ct - rt$$

An acknowledgement $hauthAK_U = \{N_U, K_{SS}\}_{K_{SS_{old}}}$ is generated responding to $hauthRQ_U$ in M1. Also, a challenge $CH'_S = \{S, N'_S\}_{K_{SS}}$ is sent to the user for mutual authentication.

M5. $U \rightarrow S : RESP_S \| A_U$

As above, the user decrypts messages to get the new session key and his temporary authenticator A_U. The new session key is used to generate the response $RESP_S = \{U, N'_S\}_{K_{SS}}$ to the CH_S and the new temporary authenticator $A_U = \{S, VT_U, K_{SS}\}_{K_{TU}}$.

3.3 Inter-Domain Authentication

The proposed inter-domain authentication takes advantages of the SNP design. All KDCs in the hierarchy share keys. This reduces the time required for querying and searching to locate the home KDC of the visiting user. A user TU_X roaming from domain X, for example, wants to access a server S_Y in a foreign domain Y. His authentication request will be sent to S_Y and then to the foreign KDC_Y. Since KDC_Y cannot authenticate the user, the authentication request will be forwarded back to the previously visited KDC_X after KDC_Y locates the KDC_X. In our proposal, a root KDC_R identifies a previously visited KDC for a foreign KDC. Once TU_X is authenticated by KDC_X, the user TU_X will receive a temporary identity TU_Y for its subsequent services in the domain Y. Fig. 3.4 illustrates the initial authentication flow for an inter-domain authentication.

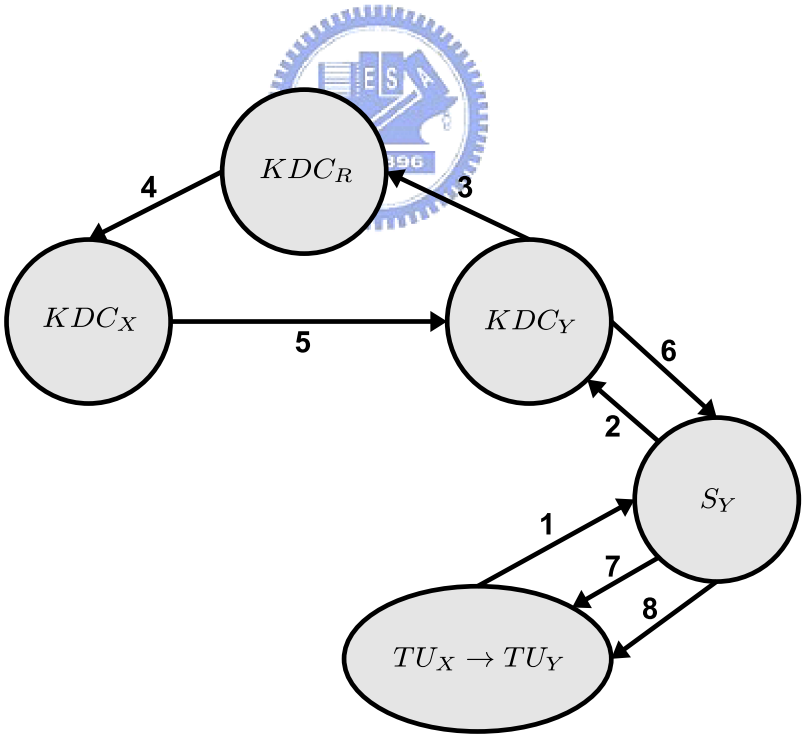


Figure 3.4: The inter-domain authentication protocol.

3.3.1 Hierarchical KDC

In the previous section, we presented an authentication protocol for authenticating users who registered in the same security domain. However, for a very large network, it is impractical for all the users to be registered in a single domain. Instead, users and servers should register with their own KDCs, which form a hierarchical structure. In such a structure, each node in the hierarchy represents a domain, where parent domains manage all their children domains. Each domain has one KDC to manage the authentication of its users and servers.

In the proposed inter-domain authentication protocol, every KDC must share a different secret key with all its ancestors to perform inter-domain authentication efficiently. Consequently, the root KDC needs a large database to store the shared keys for all descendant KDCs. Fortunately, the size of a key is small, and the root KDC is able to store all the keys.

3.3.2 Protocol Description

Similar to the intra-domain initial authentication, our approach starts with a request from the user from a foreign domain.

M1. $TU_X \rightarrow S_Y : authRQ_{TU_X}$

Assume that a user, requesting a service in a new domain Y, has a temporary user identity TU_X for its previously visited domain X. It needs to send an authentication request to the server S_Y in domain Y before accessing the desired services. The authentication request $authRQ_{TU_X}$ consists of the temporary user identity TU_X, a nonce N_{TU_X} and an encrypted message containing TU_X and N_{TU_X} using its previous temporary key OTK_{TU_X}.

$$authRQ_{TU_X} = TU_X \| N_{TU_X} \| \{TU_X, N_{TU_X}\}_{OTK_{TU_X}}$$

M2. $S_Y \rightarrow KDC_Y : authRQ_{S_Y} \| authRQ_{TU_X}$

Similarly, the server S_Y generates its authentication request $authRQ_{S_Y}$, and sends the request together with $authRQ_{TU_X}$ to its KDC_Y.

M3. $KDC_Y \rightarrow KDC_R : iauthRQ_{KDC_Y} \| authRQ_{TU_X}$

Since KDC_Y cannot authenticate the visiting user, the user authentication request is forwarded to the nearest common key distribution center (KDC_R) for KDC_X and KDC_Y. The message also includes the server identity and an authentication request from KDC_Y. The server identity is used to recognize the communication session while $iauthRQ_{KDC_Y}$ is used to authenticate the common KDC_R for KDC_Y.

$$iauthRQ_{KDC_Y} = KDC_Y \| N_{KDC_Y} \| authRQ_{TU_X}$$

The $iauthRQ_{KDC_Y}$ message contains KDC_Y's identity and a nonce N_{KDC_Y}, in plaintext.

M4. $KDC_R \rightarrow KDC_X : iauthFW_{KDC_X} \| iauthAK_{KDC_Y}$

Upon receiving the request, KDC_R sends a forwarding message to the previously visited KDC_X. The forwarding message is encrypted using the shared key of KDC_R and KDC_X and can be represented as

$$iauthFW_{KDC_X} = \{authRQ_{TU_X}, TU_Y, OTK_{TU_Y}, K_{SS}\}_{K_{KDC_X}}.$$

The forwarding message includes not only the authentication request issued by user, but also a new temporary principal name TU_Y, a new temporary user key and a new session key. In addition, KDC_R encrypts TU_Y, K_{TU_Y}, K_{SS} and the nonce in $iauthRQ_{KDC_Y}$ with the shared key of KDC_R and KDC_Y, and forwards it to KDC_X. This message is an authentication response to KDC_Y and can be represented as

$$iauthAK_{KDC_Y} = \{N_{KDC_Y}, TU_Y, OTK_{TU_Y}, K_{SS}\}_{K_{KDC_Y}}.$$

M5. $KDC_X \rightarrow KDC_Y : iauthAK_{TU_X} \| iauthAK_{KDC_Y}$

KDC_X decrypts the authentication response $iauthAK_{KDC_X}$ and gets the temporary user identity-key pair and session key. Since that KDC_X only knows the nonce and temporary user key for user TU_X, it encrypts the original nonce and new temporary user identity-key pair and session key with the pervious key OTK_{TU_X}. This is a message in response to the authentication request issued by user TU_X, the message is represented as $iauthAK_{TU_X} = \{N_{TU_X}, TU_Y, OTK_{TU_Y}, K_{SS}\}_{OTK_{TU_X}}$. The message is sent to KDC_Y together with the authentication response $iauthAK_{KDC_Y}$ from KDC_R.

M6. $KDC_Y \rightarrow S_Y : SID \| iauthAK_{S_Y} \| iauthAK_{TU_X}$

KDC_Y decrypts the authentication response $iauthAK_{KDC_Y}$, verifies the received nonce N_{KDC_Y} and extracts temporary user identity-key pair. Then, KDC_Y generates a new session identity and an authentication response to server S_Y. The response of $authRQ_{S_Y}$ can be represented as $iauthAK_{S_Y} = \{N_{S_Y}, TU_Y, K_{SS}\}_{OTK_{S_Y}}$, where $OTK_{S_Y} = Hash(S_Y, N_{S_Y}, PW_{S_Y})$ is the one-time key of S_Y.

M7. $S_Y \rightarrow TU_X : iauthAK_{TU_X} \| CH_{S_Y} \| TKT_{S_Y}$

Similar to the above description of intra-domain authentication, the server generates a challenge $CH_{S_Y} = \{S_Y, N'_{S_Y}\}_{K_{SS}}$ and a service ticket

$TKT_{S_Y} = SID \| \{TU_Y, VT_{S_Y}, K_{SS}\}_{OTK_{S_Y}}$.

M8. $TU_X \rightarrow S_Y : RESP_{S_Y} \| A_{TU_Y}$

The user generates a new temporary authenticator using its handover key HK_{TU_Y}. The authenticator $A_{TU_Y} = \{S_Y, VT_{TU_Y}, K_{SS}\}_{OTK_{TU_Y}}$ can be used for subsequent authentication. Then the user encrypts the nonce and temporary identity for the newly visited domain with the session key and sends it back as a response to CH_{S_Y}. The

response is represented as $RESP_{S_Y} = \{TU_Y, N'_{S_Y}\}_{K_{SS}}$.

Chapter 4

Software Architecture: EAP-OSNP

This chapter discusses the integration of the proposed protocol and EAP framework, which we call EAP-OSNP. EAP is a client/server protocol using different authentication methods for authenticating users requesting access to the network. There are three entities in the EAP protocol: the peer, the authenticator and the server. The EAP peer acts as a client requesting authentication and network services. The authenticator is the entity that controls the network access ports. The EAP server is capable of verifying users' credentials.

4.1 EAP Framework

Using intra-domain authentication as an example, the EAP peer builds an authentication request $authRQ_U$ and encapsulates it into EAP-OSNP messages. The EAP peer behaves exactly as a wireless client (U) in OSNP. Fig. 4.1 illustrates the EAP message flow of intra-domain initial authentication of EAP-OSNP. To comply with the EAP framework, one more message *EAP-Request* (type = S2U_HELLO) is required for initiating the EAP authentication. The *EAP-Request* and *EAP-Reponse* messages carry the OSNP authentication payload, but the messages exchanged between the EAP server and the KDC may optionally follow the EAP framework. The EAP authentication successes when the wireless client receives an *EAP-Success* message.

There are five fields in an EAP packet, as shown in Fig. 4.2. The *Code* field identifies the type of the EAP packet: 1 for request and 2 for response. The *Identifier* is a sequence number

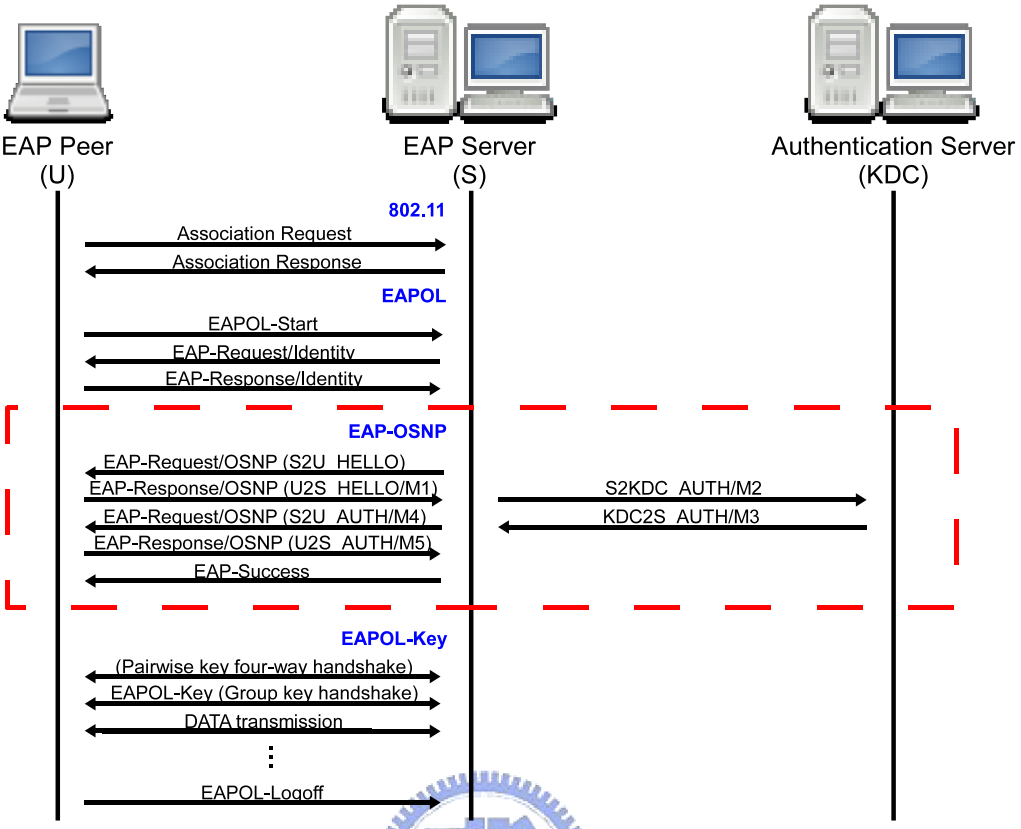


Figure 4.1: EAP-OSNP Message Flow: Intra-domain Initial Authentication.

used to match the request and response packets. The *Length* indicates the total length of the packet, in octet. The *Type* indicates the authentication method encapsulated in the EAP message; hexadecimal $0xDD$ is reserved for EAP-OSNP in our implementation. The *Type-Data* field contains the payload of EAP-OSNP message, which is composed of a *Message-Type* and an *OSNP-data*. Fig. 4.2 shows an example EAP-OSNP packet for requesting user authentication: the *Message-Type* is *U2S_HELLO* and the *OSNP-Data* is $authRQ_U$.

Table 4.1 lists the *Message-Type* and the *OSNP-Data* defined for EAP-OSNP intra-domain initial authentication.

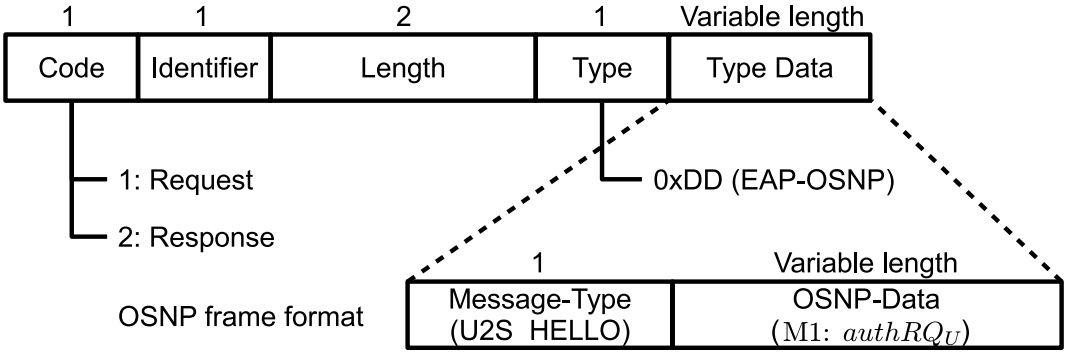


Figure 4.2: EAP-OSNP Packet Format: M1 of intra-domain initial authentication.

Table 4.1: EAP-OSNP Messages

Message-Type	OSNP-Data
S2U_HELLO	S
U2S_HELLO	M1 of Init.: $authRQ_U$
S2KDC_AUTH	M2 of Init.: $authRQ_S \| authRQ_U$
KDC2S_AUTH	M3 of Init.: $SID \| authAK_S \| authAK_U$
S2U_AUTH	M4 of Init.: $authAK_U \| CH_S \| TKT_S$
U2S_AUTH	M5 of Init.: $RESP_S \| A_U$
U2S_VT_VERIFY	M1 of Sub.: $sauthRQ_U$
S2U_VT_VERIFY_AUTH	M2 of Sub.: $sauthAK_U \| CH_S \| A_U$
U2S_SUBSEQ_AUTH	M3 of Sub.: $RESP_S$
U2S_HOREQ	M1 of Handover: $hauthRQ_U$
S2S_HOVRF	M2 of Handover: $CH_S \| hauthRQ_U$
S2S_HOVRF_ACK	M3 of Handover: $SID \| hoauthVF_{S_{old}}$
S2U_HOACK	M4 of Handover: $hauthAK_U \| CH'_S \| TKT_S \| A_{U_{old}}$
U2S_HO_AUTH	M5 of Handover: $RESP_S \| A_U$

4.2 Building Blocks

We use several software components to implement EAP-OSNP authentication system. There are four components: OSNP library, KDC server, EAP-OSNP server, and EAP-OSNP client.

- OSNP library

  The OSNP library provides the fundamental data structure, functions and application programming interfaces (APIs) required for the OSNP parts. We encapsulate OSNP

messages from this library into EAP framework. Moreover, it can be used in any other authentication programs.

- KDC server

  The KDC server is in charge of authenticating the EAP peer (the wireless client) and the EAP server. It also manages both OSNP clients' and servers' accounts and permissions, generates session keys for the EAP peers and the EAP servers.

- EAP-OSNP server

  The EAP-OSNP server implements the server protocol of OSNP, including attaching and detaching EAP-OSNP module. It is also in charge of initializing EAP-OSNP module and processing EAP-OSNP packets according to the protocol defined in Chapter 3.

- EAP-OSNP client

  The EAP-OSNP client implements the client protocol of OSNP, including receiveing and responsing the identity request from early EAP methods.

4.3 Software Modules

This section depicts the implementation of the building blocks we mentioned in the previous section.

4.3.1 OSNP Library

The OSNP library provides the data structures, functions and APIs for the EAP-OSNP entities. The OSNP library is composed of *osnp.h*, *osnp_kdc.h*, *osnp_s.h*, *osnp_u.h*, and *eaposnp_mkeys.h* as illustrated in Fig. 4.3.

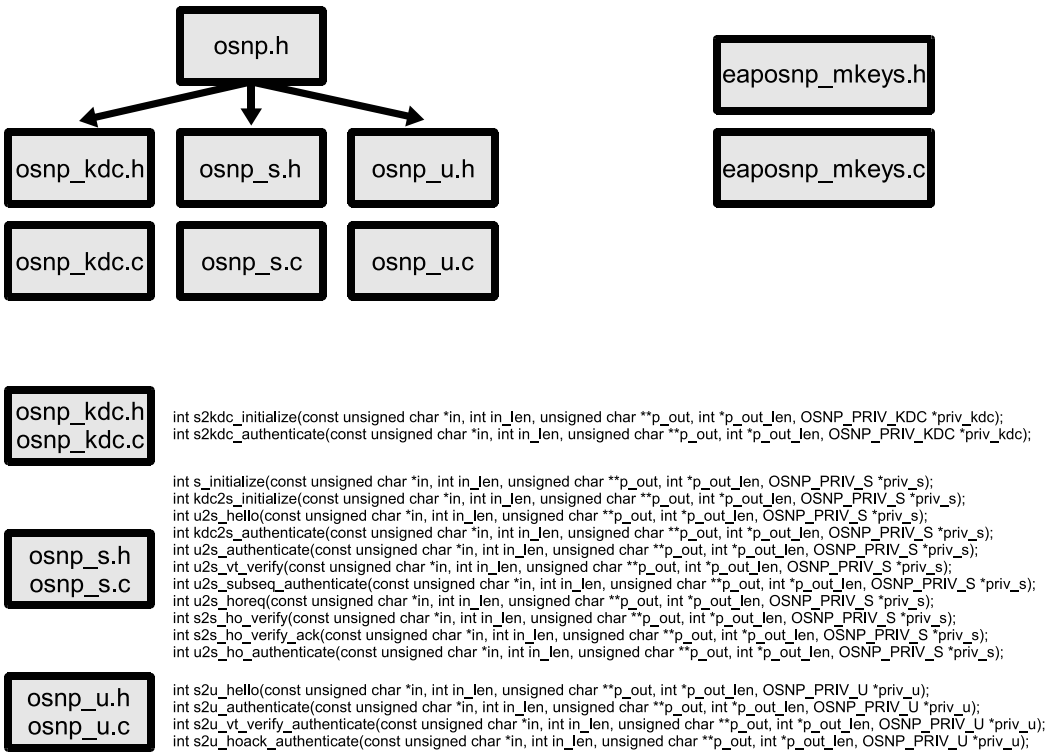


Figure 4.3: The OSNP Library.

- osnp.h

  This header file defines the major data structures, like *CIPHER_METHOD*: ciphersuite types, and *OSNP_MSG_TYPE*: message types. *OSNP_MSG_TYPE*, for example, defines the message types listed in Table 4.1.

```
typedef enum {
        OSNP_INIT,
        OSNP_S2KDC_INIT,
        OSNP_KDC2S_INIT,
        OSNP_S2U_HELLO,

        // Initial authentication
        OSNP_U2S_HELLO,
        OSNP_U2S_AUTH,
        OSNP_S2U_AUTH,
        OSNP_S2KDC_AUTH,
        OSNP_KDC2S_AUTH,

        // Subsequent authentication
        OSNP_U2S_VT_VERIFY,
        OSNP_S2U_VT_VERIFY_AUTH,
        OSNP_U2S_SUBSEQ_AUTH,
```

```
        // Handover authentication
        OSNP_U2S_HOREQ,
        OSNP_S2S_HOVRF,
        OSNP_S2S_HOVRF_ACK,
        OSNP_S2U_HOACK,
        OSNP_U2S_HO_AUTH,

        OSNP_DONE
} OSNP_MSG_TYPE;
```

It also defines the fundamental functions for all the entities, these functions are invoked in another part of OSNP library, such as

```
int otp_key_generator(unsigned char *key,
                      const char *id, unsigned char id_len,
                      const char *pw, unsigned char pw_len,
                      const unsigned char *nonce);
int osnp_encrypt(const unsigned char *in, int len,
                 ENCRYPT_DATA *p_encrypt_data,
                 const CIPHER_CTX *ctx, CIPHER_METHOD select);
int osnp_decrypt(unsigned char **p_out, int *p_len,
                 const ENCRYPT_DATA *p_encrypt_data,
                 const CIPHER_CTX *ctx, CIPHER_METHOD select);
```

- osnp_kdc.h

  This header file contains essential data structures for building a KDC daemon. It uses a structure, *OSNP_PRIV_KDC*, to store information for the APIs for the daemon. There are three APIs for the daemon: *s2kdc_initialize*, *s2kdc_authenticate*, and *sid_generator*. *s2kdc_initialize* negotiates ciphersuites and the group key for servers. *s2kdc_authenticate* authenticates both U and S in *OSNP_S2KDC_AUTH* message. *sid_generator* generates SID mentioned in Chaper 3. SID contains information about S's address timestamp when SID generated.

- osnp_s.h

  This header defines the ticket format and fundamental data structures and APIs for a server For example, *kdc2s_initialize* processes the data from *s2kdc_initialize* on the

KDC. *u2s_hello* and *u2s_authenticate* processes messages form users in intra-domain initial authentication.

- osnp_u.h

  This header file provides fundamental data structures and APIs for a user. We deal with authentication policies in function *s2u_hello*. Fig. 4.4 shows current authentication method decision. The data structure *OSNP_PRIV_U* stores the encrypted tickets obtained from Ss which have connected. First, we decide whether U can use

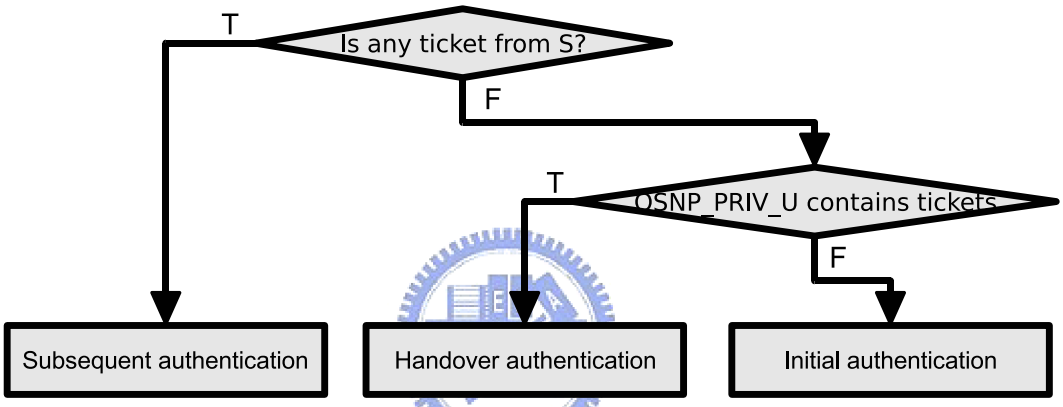


Figure 4.4: Authentication method decision of U.

subsequent authentication by searching ticket form S. Once there is no ticket from S, we will use the first ticket from ticket cache of *OSNP_PRIV_U* to process handover authentication. Otherwise, the initial authentication are performed.

- eaposnp_mkeys.h

  The header file declares that OSNP library supports key generation after OSNP authentication. For example, Pairwise Master Key (PMK) for 802.11i can be generated from OSNP by *eaposnp_gen_pmks*. The function declares as the following:

```
void eaposnp_gen_pmks(const unsigned char *key_ss, const char *prf_label,
                      const unsigned char *nonce_u, const unsigned char *nonce_s,
                      unsigned char *out);
```

The OSNP library provides fourteen authentication functions for various OSNP message types, including S2U_HELLO, U2S_HELLO, S2KDC_AUTH, KDC2S_AUTH, S2U_AUTH, U2S_AUTH, etc. Taking U2S_HELLO as an example, the server randomly generates a nonce, calculates the one-time key, refreshes the private data and then generates the output message S2KDC_AUTH after it receives U2S_HELLO message. The flow of S2KDC_AUTH is described in Fig. 4.5.

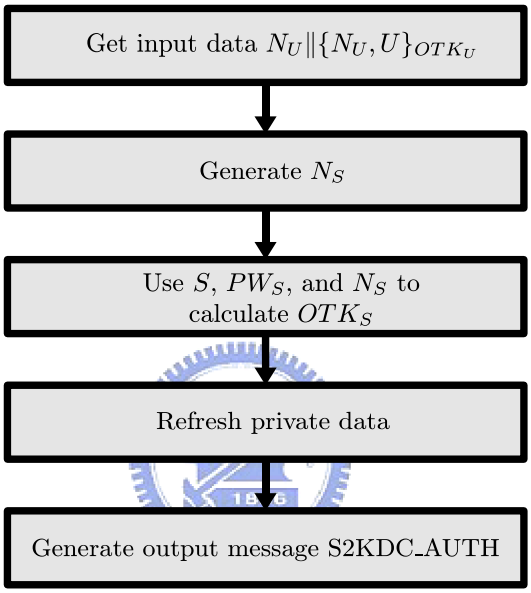
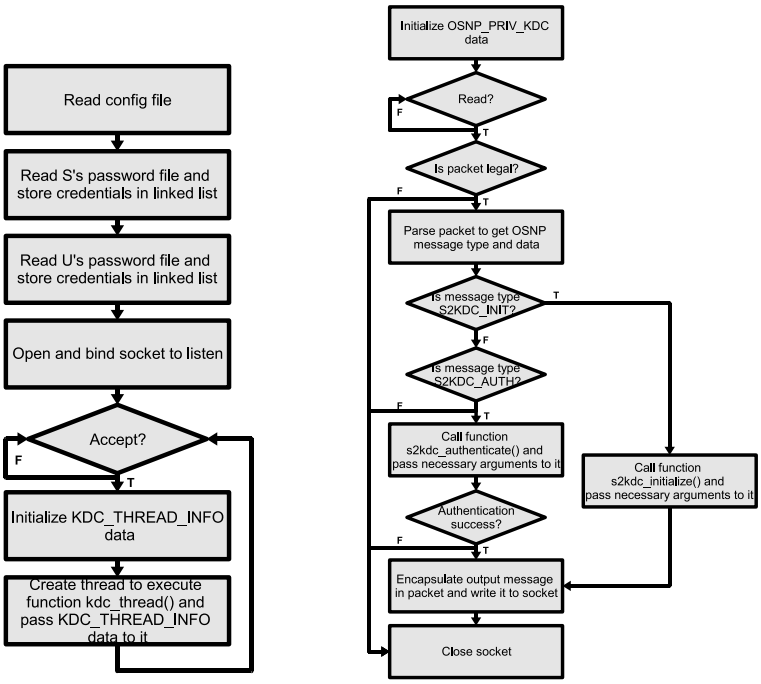


Figure 4.5: The program flow of S after receiving U2S_HELLO.

4.3.2 KDC Daemon

The KDC daemon (KDCd) module offers an administrative interface for managing both user accounts and servers permissions. The program flow of the KDCd is illustrated in Fig. 4.6. Fig. 4.6a shows the main thread of KDCd. At first, KDCd reads the account files, storing into the access lists. Then KDCd binds socket to wait for clients. After accepting the socket, the main thread generates the kdc_thread to deal with the authentication processes. The program flow of kdc_thread shows as Fig. 4.6b. Atfer checking the packet, this thread calls differnet APIs with corresponding message types.



(a) Initialization and of main thread.

(b) The kdc_thread program flow.

Figure 4.6: The KDC Daemon.

4.3.3 Authentication Server

The server module processes the EAP-OSNP, as illustrated in Fig. 4.7. The module checks message types with correct statuses and calls corresponding APIs to deal with the EAP packets. We implement the EAP-OSNP server module on FreeRADIUS [21], the open source RADIUS server with version *1.1.7* on Linux operating system.

FreeRADIUS is a modular RADIUS server. It provides several EAP authentication through AAA packets. And it is easy to add an EAP authentication sub-module by using *EAP_TYPE* structure.

```
typedef struct eap_type_t {
        const   char *name;
        int     (*attach)(CONF_SECTION *conf, void **type_data);
        int     (*initiate)(void *type_data, EAP_HANDLER *handler);
        int     (*authorize)(void *type_data, EAP_HANDLER *handler);
        int     (*authenticate)(void *type_data, EAP_HANDLER *handler);
        int     (*detach)(void *type_data);
} EAP_TYPE;
```

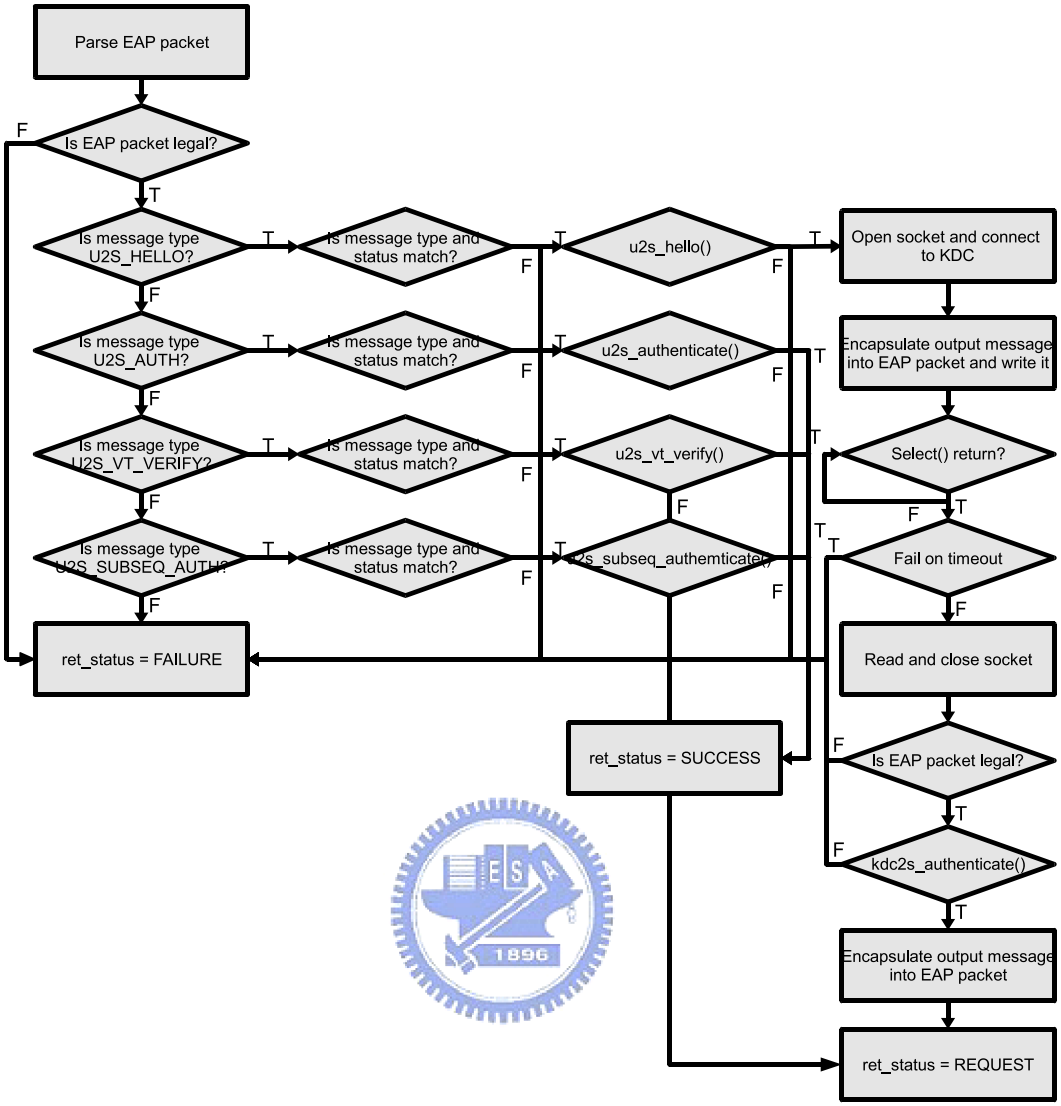


Figure 4.7: The EAP-OSNP server.

4.3.4 Wireless Client

The client module is a revision of wpa_supplicant [22], which is an open source package that implements key negotiation with a WPA authenticator and controls the roaming and IEEE 802.11 authentication/association of the WLAN driver.

4.4 Protocol Stacks

The protocol stacks of components defined in prevous sections are ilustrated in Fig. 4.8. Each role uses the OSNP library except the pass-though authenticator. The pass-though authenticator is used to convert EAP packets into AAA packets and supplies connection for EAP peers. We deploy the pass-though authenticator by using hostapd [23].

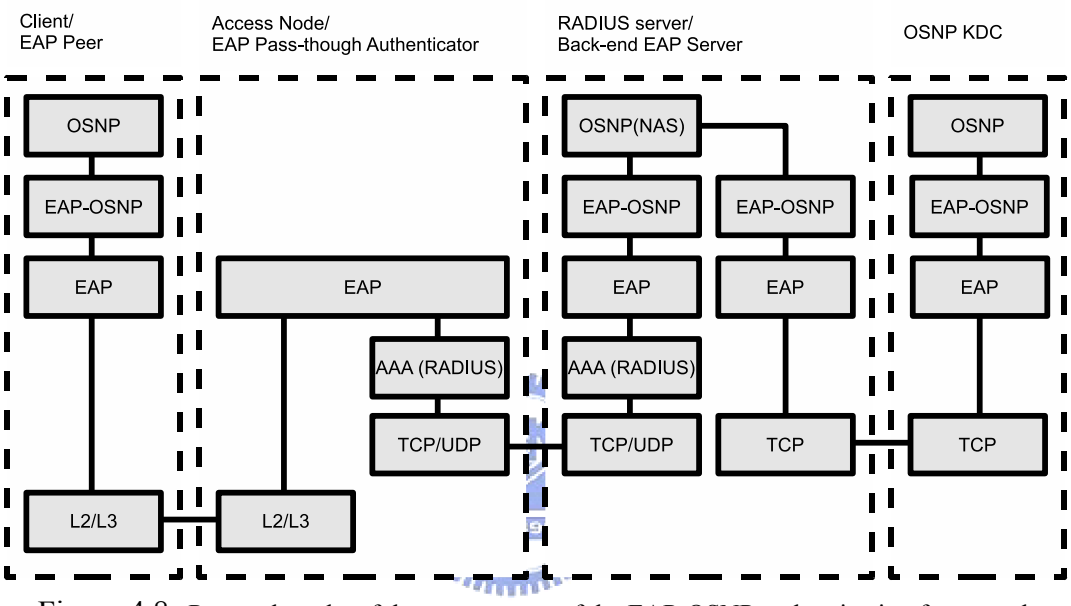


Figure 4.8: Protocol stacks of the components of the EAP-OSNP authentication framework.

Chapter 5

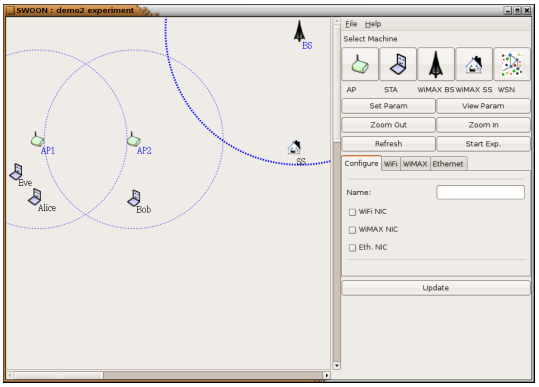
Experiments

We implement our EAP-OSNP on Linux operating system. We also use the SWOON testbed [24]to test and compare the performance of various EAP methods. This chapter describes the our experiments and results.

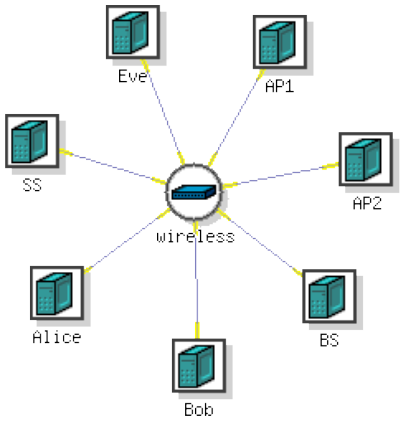
5.1 SWOON Testbed

The SWOON testbed is a comprehensive and flexible wireless testbed allowing designers to test their systems without actually building a physical test environment. We can design the network topology and deploy machines with several kinds of wireless networks, such as WiFi and WiMAX. Not like ns-2 [25], it is based on Emulab [26] [27] and DETER [28] [29], the network emulation testbed. They use NS files to describe network topology. We can set link properties between machines to shape networks, also the operating system types. Emulab deploys machines and VLANs from NS files. It reduces time of preparing experiment environments.

Besides, SWOON provides heterogeneous networks in the topology. It provides virtual wireless network through the real wired network and transformes wireless topology into wired topology connected within one switch. For example, Fig. 5.1a shows the designed topology showed in SWOON GUI. There are two parts in the GUI; WiFi part are composed of AP1, AP2, Alice, Bob, and Eve; and WiMAX part consists of BS and SS. The topology is transformed to the other wired topology shown in Fig. 5.1b.



(a) Wireless topology shows in SWOON GUI.



(b) Emulated wireless topology are connected through wired switch.

Figure 5.1: Wireless topology in SWOON GUI/real topology

SWOON uses virtual wireless devices which encapsulate packets over ether UDP broadcast packets. Therefore, it can supply four basic wireless commucation properties: broadcast, packet latency, packet loss, and eavesdropping by network shaper. We can generate desired topology by SWOON.

5.2 Experiment

We setup an experiment for validating performance of intra-domain authentications. We compare EAP-OSNP and other authentication methods: EAP-TLS, EAP-TTLS/MD5, and PEAPv0/MS-CHAPv2.

5.2.1 Topology

The real topology is shown as Fig. 5.2. The WiFi environment is composed of *s0*, *s1*, and *sta* connected on switch *wireless*. *kdc*, *s0*, and *s1* are connected with private LAN switch, *plan*. And *dst* is located on outside public network. *kdc* is the KDC server of OSNP. *s0* and *s1* act the

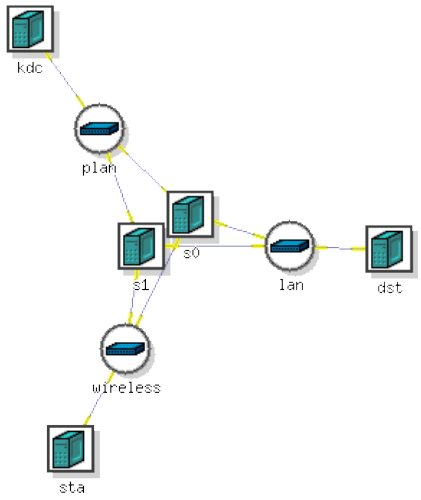


Figure 5.2: The topology of the experiment

access points and the RADIUS servers. *sta* is a wireless station, and *dst* is a outside service.

5.2.2 Software Packages

We run experiments on kernel version 2.6.20-21 with the kernel cryptographic API enabled. We deploy one wireless client, one wireless access point, one RADIUS server, and one KDCd.

- KDCd

  On the KDCd, we need to install OSNP library, and the kdcd tools. We modify the kdcd.conf for the kdc key and suitable ciphersuites. We use *osnp_useradd*, *osnp_userdel*, and *osnp_passwd* to manage accounts on this domain. There are two server accounts and one user account for all three initial authentication methods: initial, subsequent, and handover.

- RADIUS Server

  The RADIUS server acts as S, and we install OSNP library and a revision of

FreeRADIUS [21] on it. The server can use either 128-bit or 256-bit AES (Advanced Encryption Standard) ciphersuite as OSNP data encryption. The ip address and port of KDCd is 10.1.2.3 and 14000. VT_S is 3600 seconds. The OSNP section of configuration file *eap.conf* for S1 is shown as following:

```
osnp {
s_identity = eapserver1
s_passwd = gnitset
ciphersuite = "C_AES_128_CBC, C_AES_256_CBC"
kdc_ip = 10.1.2.3
kdc_port = 14000
kdc_timeout = 4
vt_interval = 3600
s_port = 14000 }
```

- Wireless Access Point

  We use hostapd [23] as our wireles access point with version *0.5.10*. The hostapd daemon is configured as a pass-though authenticator with specified ip address of RADIUS server in the hostapd configuration file.

```
ssid=wpa-osnp
ieee8021x=1
auth_server_addr=127.0.0.1
auth_server_port=1812
auth_server_shared_secret=secure
acct_server_addr=127.0.0.1
acct_server_port=1813
acct_server_shared_secret=secure
```

- Wireless Client

  We install OSNP library and a revision of wpa_supplicant [22] of version *0.5.8*. This client is open source software supporting WPA2 authentication in several platforms. In this experiment, the supplicant uses EAP-OSNP for the connected access points with SSID "wpa-osnp". Both stations and access points use 256-bit AES (Advanced Encryption Standard) as the ciphersuite of OSNP. VT_U for this supplicant is 4800 seconds. The configuration file of wpa_supplicant is like following:

```
network={
        ssid="wpa-osnp"
        key_mgmt=WPA-EAP
        eap=OSNP
        pairwise=CCMP
        group=TKIP
        identity="eapuser"
        password="testing"
        ciphersuite="C_NULL, C_AES_256_CBC"
        vt_interval=4800
        priority=20 }
```

5.2.3 Measurement

We use a packet sniffer and protocol analyzer, wireshark [30] version *1.0.0*, on the wireless client. We measure time cost and message counts during the authentication process between *EAP-Response/EAP-Identity* and *EAP-Success* as Fig. 4.1.

5.3 Results

We compare EAP-OSNP with other EAP method using for WPA2. The result shows on Table 5.1 and Fig. 5.3. The experiment runs 100 times for each authentication method. Each data point in Fig. 5.3 is the average per 10 runs. The table shows the mean for these average and message counts. All OSNP authentication methods are faster than other methods. The subsequent authentication method is faster than the initial method. However, the handover is almost equal to initial authentication, but it reduce the loading of KDC. It shows that OSNP is more efficient than other EAP methods.

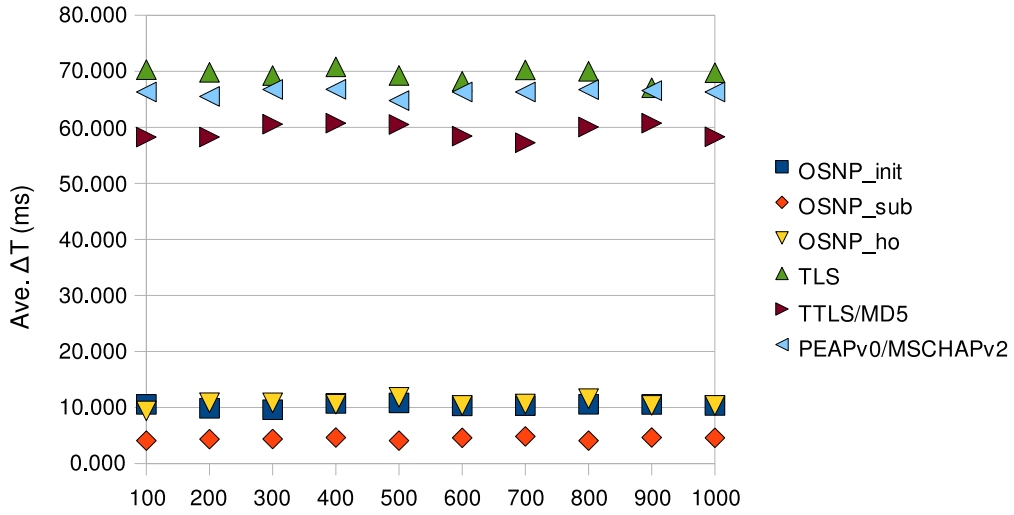


Figure 5.3: EAP methods comparison result

Table 5.1: Average Result

	OSNP initial	OSNP subsequent	OSNP handover	TLS	TTLS MD5	PEAPv0 MS-Chapv2
Δt (ms)	10.37	4.43	10.68	69.47	59.32	66.24
Authentication messages	4	4	4	14	12	18

Chapter 6

Analysis and Comparisons

6.1 Security Analysis

- Trivial Substitutions and Replay Attack

  Since all of the proposed protocols are nonce-based and every credential and ticket in our protocols contains the nonces used to verify the freshness of that credential, trivial substitutions and replays attacks can be easily detected. Similar to other nonce-based protocols, the challenger starts a timer and waits for a response. If the timer expires before receiving the response, the challenger assumes that the message is either lost or corrupted and must issue a new challenge.

- 802.1X Identity Privacy

  When an eavesdropper is listening on network traffic, the authentication process exposes the identity of the EAP peer. Even with a stolen identity, the eavesdropper still cannot login into the system without the correct one-time key. Taking intra-domain initial authentication as an example, suppose the eavesdropper stores the user's identity from previous sessions. It could then generate a forged authentication request $authRQ_U$. However, the request would fail authentication.

- Man-in-the-middle Attack

  Since all critical messages in our protocol are encrypted to prevent eavesdropping, it is

nearly impossible to modify the messages exchanged between entities. However, if an attacker A eavesdrops the communication channel between U and S, he can replace the authentication request $authRQ_U$ with $authRQ_A$. The replaced $authRQ_A$ is forwarded to the KDC together with $authRQ_S$. The attacker may be successfully authenticated by the KDC if he is a legitimate user in the system, but the man-in-the-middle attack still fails because the attacker cannot generate a correct $authAK_U$ to respond to the $authRQ_U$. Therefore, we conclude that a man-in-the-middle attack would not succeed against the OSNP protocol.

- User Impersonation Attack with Compromised Session Keys

  Since each session key is used only for a single authentication session and is discarded after authentication, an impersonation attack with a compromised session key can be prevented. In our authentication protocols, we do not rely on timestamps or temporary keys. Taking intra-domain initial authentication as an example, this kind of attack can be easily detected by a server in M3 by checking the freshness of the nonce in $authAK_S$. If the intruder substitutes N_S in M2 and replays M3, the server can still detect that M3 is simply a forged message by verifying the nonce in $authAK_S$. The intruder will be rejected even if he holds a compromised session key.

- Forward Secrecy

  Our protocol addresses forward secrecy. The disclosure of long-term secret keying material used to derive an agreed key does not compromise the secrecy of agreed keys from earlier runs [20]. In our protocol, keys are chosen randomly, and the one-time key itself is used as a key which changes with each use.

  Table 6.2 and 6.1 analyze some EAP methods, including EAP-TLS, EAP-OTP, EAP-Kerberos and EAP-OSNP, and compares their characteristics and capability against

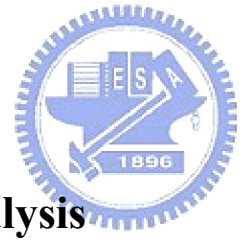
Table 6.1: Authentication Analysis

EAP Methods	TLS	OTP	Kerberos	OSNP
Server Authentication	Certificate	None	Password	OTK
Client Authentication	Certificate	OTK	Password	OTK
Mutual Authentication	Yes	No	Yes	Yes

Table 6.2: Security Analysis

EAP Methods	TLS	OTP	Kerberos	OSNP
Replay Attack	Yes	Yes	Yes	Yes
Dictionary Attack	Yes	No	No	No
Brute-Force Attack	No	Yes	No	Yes
Identity Privacy Protection	No	No	Yes	No
Man-in-the-middle Attack	Yes	No	Yes	Yes
User Impersonation Attack	No	No	Yes	Yes
Forward Secrecy	No	Yes	Yes	Yes

attacks.



6.2 Performance Analysis

In Sec. 5.3, we show the performance of EAP-OSNP. Furthermore, this section presents the performance of OSNP. Tables 6.3 and 6.4 show the performance of Kerberos and OSNP, in terms of computation, communication and storage.

Table 6.4 shows the number of messages for mutual authentication and the number of messages submitted by a user. OSNP requires a constant number of messages independent of the number of KDCs between the user's visited domain and home domain. This reduces the time required for roaming from one domain to another. Compared with the Kerberos protocol, OSNP requires only two messages on the user side. This is feasible and practical for mobile networks with low data rates and bandwidth. It is also good for battery-powered mobile devices.

Table 6.3: Performance Comparison: Computation Cost

Auth	Operation	KDC		S_{old}		S		U	
		K	O	K	O	K	O	K	O
Init	Random	2	2	-	-	0	2	2	1
	Hash	4	2	-	-	3	1	4	1
	En/decrypt	6	5	-	-	5	5	5	5
Sub	Random	-	-	-	-	-	2	-	1
	Hash	-	-	-	-	-	1	-	0
	En/decrypt	-	-	-	-	-	4	-	4
Ho	Random	-	-	-	0	-	3	-	1
	Hash	-	-	-	1	-	1	-	0
	En/decrypt	-	-	-	3	-	5	-	5

Table 6.4: Performance Comparison: Communication and Storage Costs

	#Msg for Mutual Auth	#Msg from User	Type of Trust	#Shared Keys	Mobility Support
Krb	$2m+4$	$m+2$	P&H	$O(N)$	No
OSNP	8	2	P&H	$O(N)$	Yes

m: number of KDCs between the user's visited domain and home domain

N: number of domains

P&H Peer and Hierarchical

Table 6.4 also compares the number of shared keys among these protocols. Consider a hierarchy with N domains. The number of shared keys in OSNP is proportional to the number of domains, which is the same as the number of shared keys in Kerberos V5.

6.3 OSNP Logic Proof

We use BAN logic [13] and its enhancement [14] to explain why our protocol can reach the goals of mutual authentication for initial and subsequent authentication of intra-domain authentication. The intra-domain handover is a derivation of initial authentication, replacing

KDC with the previous server and the inter-domain authentication is an extension of the initial authentication; the security of these two types of authentication is guaranteed by that of the initial authentication.

The BAN logic states that the mutual authentication is complete between two parties A and B, if there is a K such that

$$A \textbf{ believes } A \overset{K}{\longleftrightarrow} B,$$

$$B \textbf{ believes } A \overset{K}{\longleftrightarrow} B,$$

$$A \textbf{ believes } B \textbf{ believes } A \overset{K}{\longleftrightarrow} B,$$

$$B \textbf{ believes } A \textbf{ believes } A \overset{K}{\longleftrightarrow} B.$$

6.3.1 Initial Authentication

The objectives of the initial authentication are to prove: the presence of both parties to each other, and the receipt of a ticket and a session key at the user side. Assume that

$$U \textbf{ believes } U \overset{OTK_U}{\longleftrightarrow} KDC, \text{ and} \tag{6.1}$$

$$S \textbf{ believes } S \overset{OTK_S}{\longleftrightarrow} KDC. \tag{6.2}$$

The proof is given in two parts: to authenticate S by U, and to authenticate U by S.

For the first part, since U receives $authAK_U$, CH_S and TKT_S in M4, he can decrypt $authAK_C$ and get the session key K_{SS}. By applying annotation rule and formula 6.1, we

obtain

$$U \textbf{ believes } U \overset{OTK_U}{\longleftrightarrow} KDC,$$

$$U \textbf{ sees } \{N_U, S, U \overset{K_{SS}}{\longleftrightarrow} S\}OTK_U, \text{ and}$$

$$U \textbf{ believes } KDC \textbf{ said } (N_U, S, U \overset{K_{SS}}{\longleftrightarrow} S).$$

Since that N_U is generated by U, we have the following hypothesis:

$$U \textbf{ believes fresh } (N_U, S, U \overset{K_{ss}}{\longleftrightarrow} S).$$

The nonce-verification rule applies and yields

$$U \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S.$$

By decrypting the CH_S, U verifies the server identity. Similarly, we obtain

$$U \textbf{ sees } \{S, N'_S\}K_{SS},$$

$$U \textbf{ believes } S \textbf{ said } (S, N'_S, U \overset{K_{SS}}{\longleftrightarrow} S),$$

$$U \textbf{ believes } S \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S.$$

The second part is proved by M3 and M5. S receives $authAK_S$ in M3, and he can decrypt the token and extract the session key K_{SS}. Then, S decrypts $RESP_S$ using K_{SS} to get N'_S. Similarly, applying the annotation, the message-meaning, jurisdiction rules, and formula 6.2, we obtain

$$S \textbf{ believes } S \overset{OTK_S}{\longleftrightarrow} KDC,$$

$$S \textbf{ sees } \{N_S, U, U \overset{K_{SS}}{\longleftrightarrow} S\}OTK_S, \text{ and}$$

$$S \textbf{ believes } KDC \textbf{ said } (N_S, U, U \overset{K_{ss}}{\longleftrightarrow} S).$$

Since that N_S is generated by S, we have the following hypothesis:

$$S \textbf{ believes fresh } (N_S, U, U \overset{K_{SS}}{\longleftrightarrow} S), \text{ and}$$

$$S \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S.$$

Similarly, N'_S is generated by S, we apply the nonce-verification rule and jurisdiction rule and obtain

$$S \textbf{ sees } \{N'_S\}K_{SS},$$

$$S \textbf{ believes fresh } (N'_S, U \overset{K_{SS}}{\longleftrightarrow} S),$$

$$S \textbf{ believes } U \textbf{ said } (N'_S, U \overset{K_{SS}}{\longleftrightarrow} S), \text{ and}$$

$$S \textbf{ believes } U \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S.$$

It proves that our initial authentication can achieve the following goals at the end of the authentication round:

$$U \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S,$$

$$S \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S,$$

$$U \textbf{ believes } S \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S, \text{ and}$$

$$S \textbf{ believes } U \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S$$

□

6.3.2 Subsequent Authentication

In our subsequent authentication, S decrypts TKT_S and extracts U, VT_S and K_{SS}. After checking the validation of VT_S, K_{SS} is still validate. We can apply the above formal rules, and

obtain

$$U \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S,$$

$$U \textbf{ sees } \{N_U, U \overset{K'_{SS}}{\longleftrightarrow} S\}K_{SS},$$

$$U \textbf{ believes } S \textbf{ said } (N_U, U \overset{K'_{SS}}{\longleftrightarrow} S),$$

$$U \textbf{ believes fresh } (N_U, U, U \overset{K'_{SS}}{\longleftrightarrow} S), \text{ and}$$

$$U \textbf{ believes } S \textbf{ believes } U \overset{K'_{SS}}{\longleftrightarrow} S.$$

For S, we also prove that

$$S \textbf{ believes } S \overset{OTK_S}{\longleftrightarrow} KDC,$$

$$S \textbf{ sees } \{U, VT_S, U \overset{K_{SS}}{\longleftrightarrow} S\}OTK_S,$$

$$S \textbf{ believes } KDC \textbf{ said } (U, VT_S, U \overset{K_{SS}}{\longleftrightarrow} S).$$

So,

$$S \textbf{ believes } U \overset{K_{SS}}{\longleftrightarrow} S.$$

After receiving $RESP_S$, which contains a nonce N_S and a new session key K'_{SS} generated by S, we obtain

$$S \textbf{ sees } \{N_S\}K'_{SS},$$

$$S \textbf{ believes fresh } (N_S, U \overset{K'_{SS}}{\longleftrightarrow} S),$$

$$S \textbf{ believes } U \textbf{ said } (N_S, U \overset{K'_{SS}}{\longleftrightarrow} S),$$

$$S \textbf{ believes } U \textbf{ believes } U \overset{K'_{SS}}{\longleftrightarrow} S.$$

This proves that our subsequent authentication can achieve the above goals. □

Chapter 7

Conclusion

In this paper, we integrated one-time keys with a nonce-based authentication protocol, which efficiently supports initial, subsequent and handover authentication. In design part, our protocol requires five messages for initial authentication; three for subsequent authentication and five for handover authentication. Although five messages are required for handover authentication, no KDC is involved in authenticating the roaming user. Then, we extended the intra-domain authentication protocol to an inter-domain authentication protocol, which requires eight messages for mutual authentication, regardless of the number of hops between the visited and home domains. In all our authentication protocols, only two messages are sent by the user. Such a design is very feasible and practical for a mobile network with limited bandwidth and for those battery-powered mobile devices.

Since KDCs are transparent to users in OSNP, only registered servers can communicate with KDCs directly. This architecture is suitable for the current mobile network, where mobile devices only need to connect to a visited server for authentication, without knowing the location of KDCs.

In implementation part, the intra-domain authentication methods are completely implemented with modular design. Using the revision of open source authentication applications is convenient for integrating our protocol with other authentication protocols, also convenient for comparisons.

References

[1] IEEE, "IEEE Standard for Local and metropolitan area networks Port-Based Network Access Control," 2004, pp. 1--169.

[2] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748 (Proposed Standard), Jun. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3748.txt

[3] D. Stanley, J. Walker, and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs," RFC 4017 (Informational), Mar. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4017.txt

[4] IEEE, "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 (Includes IEEE Std 802.11, 1999 Edition; IEEE Std 802.11a.-1999; IEEE Std 802.11b.-1999; IEEE Std 802.11b.-1999/Cor 1-2001; and IEEE Std 802.11d.-2001)," 2005, pp. 1--721.

[5] B. Aboba and D. Simon, "PPP EAP TLS Authentication Protocol," RFC 2716 (Experimental), Oct. 1999, obsoleted by RFC 5216. [Online]. Available: http://www.ietf.org/rfc/rfc2716.txt

[6] D. Simon, B. Aboba, and R. Hurst, "The EAP-TLS Authentication Protocol," RFC 5216 (Proposed Standard), Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5216.txt

[7] N. Cam-Winget, D. McGrew, J. Salowey, and H. Zhou, "The Flexible Authentication via

Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)," RFC 4851 (Informational), May 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4851.txt

[8] K.-H. Baek, S. W. Smith, and D. Kotz, "A Survey of WPA and 802.11i RSN Authentication Protocols," Dept. of Computer Science, Dartmouth College, Hanover, NH, Tech. Rep. TR2004-524, November 2004. [Online]. Available: http://www.cs.dartmouth.edu/~dfk/papers/baek-survey-tr.pdf

[9] C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The Kerberos Network Authentication Service (V5)," RFC 4120 (Proposed Standard), Jul. 2005, updated by RFCs 4537, 5021. [Online]. Available: http://www.ietf.org/rfc/rfc4120.txt

[10] S. Zrelli and Y. Shinoda, "Specifying Kerberos over EAP: Towards an integrated network access and Kerberos single sign-on process," in *Advanced Information Networking and Applications*, 2007, pp. 490--497, AINA '07. 21st International Conference.

[11] *A Real-World Analysis of Kerberos Password Security*, 1999. [Online]. Available: citeseer.ist.psu.edu/wu99realworld.html

[12] S. M. Bellovin and M. Merritt, "Limitations of the kerberos authentication system," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 5, pp. 119--132, 1990.

[13] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18--36, 1990.

[14] S.-P. Shieh, F.-S. Ho, and Y.-L. Huang, "An Efficient Authentication Protocol for Mobile Networks," *J. Inf. Sci. Eng.*, vol. 15, no. 4, pp. 505--520, 1999.

[15] C. Xiao-rong, F. Qi-yuan, D. Chao, and Z. Ming-quan, "Research and realization of authentication technique based on OTP and Kerberos," in *High-Performance Computing*

in Asia-Pacific Region, 2005. Proceedings. Eighth International Conference on, Nov./ Dec. 2005.

[16] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), Jun. 2000, updated by RFCs 2868, 3575, 5080. [Online]. Available: http://www.ietf.org/rfc/rfc2865.txt

[17] C. Rigney, W. Willats, and P. Calhoun, "RADIUS Extensions," RFC 2869 (Informational), Jun. 2000, updated by RFCs 3579, 5080. [Online]. Available: http://www.ietf.org/rfc/rfc2869.txt

[18] B. Aboba and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)," RFC 3579 (Informational), Sep. 2003, updated by RFC 5080. [Online]. Available: http://www.ietf.org/rfc/rfc3579.txt

[19] D. Nelson and A. DeKok, "Common Remote Authentication Dial In User Service (RADIUS) Implementation Issues and Suggested Fixes," RFC 5080 (Proposed Standard), Dec. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5080.txt

[20] Y. Ohba, S. Das, and A. Dutta, "Kerberized handover keying: a media-independent handover key management architecture," in *MobiArch '07: Proceedings of first ACM/ IEEE international workshop on Mobility in the evolving internet architecture*. New York, NY, USA: ACM, 2007, pp. 1--7.

[21] "FreeRADIUS -- The world's most popular RADIUS Server." http://www.freeradius.org/.

[22] "Linux WPA/WPA2/IEEE 802.1X Supplicant," http://hostap.epitest.fi/wpa_supplicant/.

[23] "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," http://hostap.epitest.fi/hostapd/.

[24] Y. L. Huang, H. Y. L. J. D. Tygar, L. Y. Yeh, H. Y. Tsai, K. Sklower, S. P. Shieh, C. C. Wu, P. H. Lu, S. Y. Chien, Z. S. Lin, L. W. Hsu, C. W. Hsu, C. T. Hsu, Y. C. Wu, and M. S. Leong, "SWOON: A Testbed for Secure Wireless Overlay Networks," in *CSET' 08*, 2008.

[25] "The Network Simulator - ns-2," http://www.isi.edu/nsnam/ns/.

[26] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255--270, 2002.

[27] "Emulab - Network Emulation Testbed," http://www.emulab.net/.

[28] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with deter: a testbed for security research," *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. 2nd International Conference on*, pp. 10 pp.--, March 2006.

[29] "cyber-cyber-Dcyber-DEfense Technology Experimental Research laboratory Testbed," http://www.isi.edu/deter/.

[30] "Wireshark: network protocol analyzer." http://www.wireshark.org/.