Design and Implementation of WIRE1x

Yu-Ping Wang[2], Jyh-Cheng Chen[1,2], and Yi-Wen Liu[1]
[1]Department of Computer Science
[2]Institute of Communications Engineering
National Tsing Hua University
Hsinchu, Taiwan
Email: {ypwang,jcchen,ywliu}@wire.cs.nthu.edu.tw

November 11, 2004

Abstract

This paper presents the design and implementation of WIRE1x. The WIRE1x is an open-source implementation of IEEE 802.1x client (supplicant) developed by the Wireless Internet Research & Engineering (WIRE) Laboratory. The IEEE 802.1x standard defines a port-based network access control to authenticate and authorize devices interconnected by various IEEE 802 LANs. The IEEE 802.11i also incorporates 802.1x as its authentication solution for IEEE 802.11 wireless LANs. The motivation for the development of WIRE1x is to help the deployment of IEEE 802.1x to secure wireless WLANs. It would also achieve seamless roaming among different wireless LANs if the new IEEE standards are widely-deployed. The WIRE1x has been practically used on the wireless LANs deployed at the National Tsing Hua University and many other places. This paper illustrates the software architecture and design principles of WIRE1x so people could comprehend the source code of WIRE1x easily.

Keywords: Authentication, Extensible Authentication Protocol (EAP), IEEE 802.1x, IEEE 802.11i, Wireless LANs, Security

1 Introduction

Wireless local area network (WLAN) is more and more prevailing in these years. It, however, has been widely reported that security has been a weakness of the current IEEE 802.11 standards. The Task Group I of IEEE P802.11 Working Group is defining 802.11i [1, 2] to enhance security in current 802.11 standards. The 802.11i incorporates 802.1x [3] as the authentication solution for 802.11 wireless LANs. The IEEE 802.1x standard is a port-based network access control to authenticate and authorize devices interconnected by various IEEE 802 LANs. The IEEE 802.11i is expected to play a critical role in improving the overall security of current and future WLANs.

The IEEE 802.1x standard has been well-defined. Currently, many manufactures of 802.11 Access Point (AP) also support 802.1x. The 802.1x-capable APs have been deployed in many universities, organizations, and companies. To be authenticated by using 802.1x, end users also need to be 802.1x-capable. Unless 802.1x is embedded in the Operating System (OS), users generally will need to install a 802.1x client in order to access to the network. Open1x [4], an open-source implementation of 802.1x, supports Mac OS X, FreeBSD, OpenBSD, and Linux. Many users, however, are using MS Windows. They need a 802.1x client software to access to the 802.1x-based LANs. We therefore develop the *WIRE1x* to support various versions of MS Windows. As the name suggested, WIRE1x is an open-source implementation of IEEE 802.1x client (supplicant)* developed by the Wireless Internet Research & Engineering (WIRE) Laboratory†.

Currently, WIRE1x supports MS Windows XP (without service pack and with service pack 1), Windows 2000, Windows ME, and Windows 98. It provides EAP-MD5 [5], EAP-TLS [6], EAP-TTLS [7], and PEAP [8]. WIRE1x works with freeRADIUS [9]. The implementation of WIRE1x is based on Open1x, and is developed by using MS Visual C++. It utilizes libraries of WinPcap [10], Libnet [11], and OpenSSL [12]. Both source code and executable code of WIRE1x can be downloaded freely from *http://wire.cs.nthu.edu.tw/wire1x/*. This paper presents all components of WIRE1x exhaustively. The objective of this paper is to share our experience on implementing 802.1x supplicant. By reading this paper, one should easily comprehend the source code of WIRE1x.

1.1 Motivation and Contribution

Currently, the National Tsing Hua University (NTHU) has deployed WLANs and is using IEEE 802.1x and RADIUS [13] to authenticate users. Most users at NTHU are using MS Windows. However, we perceive that none of the IEEE 802.1x client software meet all demands for users of MS Windows in the marketplace up to now. Therefore, a major motivation for the development of WIRE1x is to solve the existing problem and then help IEEE 802.1x to be deployed rapidly. It would also expedite the objective of seamless roaming among different wireless LANs if the new IEEE standards are widely-deployed. The contributions include:

- WIRE1x is a free as well as open-source software. We believe open source is essential for any security

*Supplicant is a terminology defined in 802.1x which will be described in Section 2.

†http://wire.cs.nthu.edu.tw/

related software because it can be examined as you wish.

- WIRE1x provides various EAP (Extensible Authentication Protocol) [14] based authentication methods, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP.
- WIRE1x can work with various versions of MS Windows, including Windows XP, 2000, ME, and 98.
- WIRE1x works well with various types of WLAN cards. Please see the webpage at http://wire.cs.nthu.edu.tw/wire1x/compatible.htm.

We believe that WIRE1x is a good choice for people eager for IEEE 802.1x client software. Since WIRE1x was released on June 18, 2003, the website of WIRE1x has been visited for more than 15,000 times. There have been more than 2,100 downloads of source code and 3,800 downloads of executable code up to November 2004.

The rest of the paper is organized as follows. Section 2 provides a brief overview of IEEE 802.11i and IEEE 802.1x. Section 3 describes other available implementations of 802.1x. Section 4 discusses the design and implementation of WIRE1x in details. Section 5 presents the real-world applications of WIRE1x. Section 6 summarizes the paper.

2 Overview of IEEE 802.11i

This section briefly reviews WLAN security and IEEE 802.11i. For more details, please refer to [1, 2].

The IEEE 802.11 standard is one of the most widely adopted standards for broadband wireless Internet access. However, the security consideration over wireless environment is more complicated than that in wired environment. Due to the wide-open nature of wireless radio, the network is more vulnerable. The original IEEE 802.11 standard has defined the following two basic security mechanisms for securing the access to IEEE 802.11 networks: (1) entity authentication including *open system* authentication and *shared key* authentication, and (2) Wired Equivalent Privacy (WEP). Nevertheless they are all proved to be vulnerable.

To enhance the security in IEEE 802.11, the IEEE 802.11i has been proposed. In addition to introducing *key management and establishment*, it also defines *encryption* and *authentication* improvements. In order to manage security keys automatically, the IEEE 802.11i has defined key management and establishment algorithms. As conventional WEP is known to be vulnerable, the IEEE 802.11i has specified enhanced encryption algorithms to provide stronger privacy. The IEEE 802.11i also incorporates IEEE 802.1x as its authentication enhancement. The following sections briefly discuss *Encryption Algorithm* and *802.1x*.

2.1 Encryption Algorithm

Two algorithms, *Temporal Key Integrity Protocol (TKIP)* and *Counter mode with CBC-MAC Protocol (CCMP)* have been proposed. Both TKIP and CCMP provide enhanced data integrity and confidentiality over WEP. TKIP, initially referred to as WEP2, is also based on RC4 encryption as that in WEP. It however

is implemented in a different way that addresses the vulnerabilities of WEP. TKIP can be adopted into the older IEEE 802.11 products through relatively simple firmware upgrade. This is especially favorable for vendors. TKIP is optional in IEEE 802.11i.

In addition to TKIP which is considered as a short-term solution for WLAN security, the IEEE 802.11i has also defined CCMP as a long-term solution. The CCMP employs the stronger encryption of Advanced Encryption Algorithm (AES) [15] which uses the CCM mode [16] with a 128-bit key and a 128-bit block size of operation. The CCM mode combines Counter-Mode (CTR) and Cipher Block Chaining Message Authentication Code (CBC-MAC). The CTR is used to encrypt the payload and the *Message Integrity Code (MIC)* to provide confidentiality service. The CBC-MAC computes the MIC to provide authentication and integrity services. Although the CCMP could provide much stronger security services, it requires additional hardware (co-processor) to improve encryption performance. Unlike TKIP, CCMP is mandatory in IEEE 802.11i.

2.2 802.1x

The IEEE 802.1x defines a mechanism for port-based network access control. It is based upon EAP to provide compatible authentication and authorization mechanisms for devices interconnected by IEEE 802 LANs. As depicted in Fig. 1, there are three main components in the IEEE 802.1x authentication system. In a WLAN, *supplicant* usually is a Mobile Node (MN). AP usually represents an *authenticator*. Authentication, Authorization, and Accounting (AAA) server such as RADIUS server is the *authentication server*. The *port* in 802.1x represents the association between a MN and an AP. Both supplicant and authenticator have a *Port Access Entity (PAE)* that operates the algorithms and protocols associated with the authentication mechanisms. In Fig. 1, the authenticator's *controlled port* is in unauthorized state. That is, the port is open. Messages will be directed only to the *Authenticator PAE*, which will further direct 802.1x messages to the authentication server. The authenticator PAE will close the controlled port after the supplicant is authenticated successfully. Thus, the supplicant is able to access to other services through the controlled port.

Based upon EAP, the IEEE 802.1x standard can use a number of authentication mechanisms. The authentication mechanisms are outside the scope of the IEEE 802.1x. Many authentication mechanisms such as Message Digest 5 (MD5), Transport Layer Security (TLS), Tunneled TLS (TTLS), Protected Extensible Authentication Protocol (PEAP), and Lightweight Extensible Authentication Protocol (LEAP) [17] could be used. The IEEE 802.1x also defines EAP over LANs (EAPOL) to encapsulate EAP messages between the supplicant and the authenticator. The authenticator PAE relays all EAP messages between the supplicant and the authentication server. The IEEE 802.1x is utilized to enforce the use of specific authentication mechanism and to route authentication messages properly, while the authentication mechanisms define the actual authentication exchanges that take place.

Fig. 2 shows a typical 802.1x message exchange. The associated PAE state transitions in supplicant and authenticator are specified in Fig. 3 and Fig. 4, respectively. In Fig. 2, the digits in rectangles refer to the

supplicant PAE states in Fig. 3, and digits in circles refer to the authenticator PAE states in Fig. 4. In Fig. 2, RADIUS is served as the authentication server. This does not limit the use of other AAA servers such as Diameter [18] as the authentication server. Detailed discussion of Figs. 2–4 can be found in [2].

3 Related Work

This section discusses related implementations of 802.1x in the marketplace.

Table 1 lists most of commercial products of 802.1x supplicant. The supported operating systems and EAP authentication methods are also specified. Although some of them provide pre-compiled binary code for non-commercial use or free trial, basically they are not freeware. In addition, the source code are not open to public either. Table 2 lists the open-source implementations of 802.1x supplicant. The Open1x is designed to work with Linux although its earlier versions also supported BSD and Mac OS. Because many users are eager for a free software of 802.1x supplicant to work with various versions of MS Windows, we therefore have developed WIRE1x. Both Open1x and WIRE1x are licensed under the BSD License and GNU General Public License (GPL). In addition to Open1x and WIRE1x, a newly initiated project called wEAP [19] is writing open-source plug-ins for MS Windows 2000 and XP. Unlike wEAP, the WIRE1x is an application program for various versions of MS Windows.

Table 3 lists the authentication servers and the EAP they support. Except freeRADIUS, all of them are commercial products. As indicated in Table 3, the freeRADIUS supports many authentication mechanisms. Others only support some of them. A user would not be authenticated if there is no common authentication mechanism supported in both supplicant and authentication server. This will limit the roaming capability significantly.

The supplicant software is indispensable to IEEE 802.1x standard. We observe that the success of 802.1x would greatly depend on end users. We believe that a free 802.1x supplicant software which works with various versions of MS Windows and supports most of EAP authentication methods will boost the deployment of 802.1x, and thus 802.11i. Therefore, we have developed WIRE1x and hope that most users could access to WLANs with a more secure way.

4 WIRE1x

The software architecture of WIRE1x can roughly be divided into three components as illustrated in Fig. 5: (1) supplicant PAE state machine, (2) EAP and authentication mechanisms, and (3) WinPcap, Libnet, and OpenSSL. The *supplicant PAE state machine* follows the specification defined in the IEEE 802.1x for a supplicant. The state machine is also depicted in Fig. 3. As the name suggested, the *EAP and authentication mechanisms* support various authentication mechanisms in EAP including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP. The *WinPcap* [10] and *Libnet* [11] are two open-source libraries which are responsible for capturing and writing packets to and from the data link layer. Additionally, the *OpenSSL* [12] is another

open-source library which is used only for TLS-based authentication methods. The following sections discuss the three main components in details.

4.1 Supplicant PAE State Machine

The supplicant PAE state machine is the core of any implementation of 802.1x supplicant. It specifies the behavior of the supplicant and interacts with the authenticator. Again, the supplicant PAE state machine is specified in Fig. 3 and the source code can be downloaded from *http://wire.cs.nthu.edu.tw/wire1x/*. In WIRE1x, roughly speaking, it is implemented in four files: dot1x_globals.cpp, eap.cpp, eapol.cpp, and os_generic.cpp. All variables of state machine are defined in dot1x_globals.h. Additionally, the EAP *code field* and *type field* specified in RFC 2284 [14] are defined in eap.h. The eap.cpp is responsible for building the response frames and decoding the EAP packets. Moreover, EAPOL header and Ethernet header are defined in eapol.h. The eapol.cpp is responsible for starting EAPOL process, performing necessary PAE state actions, transiting to proper states, decoding the EAPOL packets, and transmitting EAPOL frames. Furthermore, os_frame_funcs.h comprises get_frame(), send_frame(), more_frames(), and so on. In os_generic.cpp, get_frame() employs pcap_dispatch() to capture EAP frames. The send_frame() employs libnet_write_link() to send EAP frames.

4.2 EAP and Authentication Mechanisms

This section introduces the Extensible Authentication Protocol (EAP). In addition, we also use the message exchanges depicted in Fig. 2 and the associated state machines shown in Fig. 3 and Fig. 4 to demonstrate a typical authentication procedure of WIRE1x. Additionally, we provide detailed description of several EAP authentication methods implemented in WIRE1x.

4.2.1 Overview

EAP [14] is a general protocol for authentication. It supports multiple authentication methods, such as token card, Kerberos [20], one-time password, certificate, public key authentication, and smart card. Fig. 6 shows that there can be many different authentication methods in the *Authentication Layer*. The authentication methods are based upon EAP. Any new authentication mechanisms can be added easily. The WIRE1x is expected to be versatile in authentication mechanisms. It has been implemented to support the most common authentication methods, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP.

Next, we present a typical authentication procedure of WIRE1x by using Figs. 2–4.

1. User opens and selects the device to be authenticated by pcap_findalldevs(). MN starts to associate with AP. Both of MN and AP then will transition to the CONNECTING state.

2. MN sends an EAPOL-start frame by libnet_write_link() to the AP to initialize the authentication process.

3. When the AP receives EAPOL-Start, it will reply EAP-Request/Identity to obtain the MN's identity. When the MN captures the EAP frame by pcap_dispatch(), the EAP frame is parsed by eap_decode_packet() and eapol_decode_packet() located in eap.cpp and eapol.cpp, respectively. Moreover, according to the result determined by eap_decode_packet() and eapol_decode_packet(), the supplicant PAE state machine transits to ACQUIRED state if the request is received successfully.

4. The MN sends back EAP-Response/Identity containing MN's identity to the authenticator. Subsequently, the authenticator and authentication server will perform necessary message exchanges.

5. When the MN receives EAP-Request/Auth which contains RADIUS-Access-Challenge, the supplicant PAE state machine transits to the AUTHENTICATING state. The MN replies an EAP-Response/Auth to the authenticator in which the RADIUS-Access-Request is encapsulated.

6. Based on the result of the EAP authentication method, the RADIUS server decides whether to authorize the user or not. If the user is authorized, the supplicant captures the EAP-Success and the supplicant PAE state machine transits to the AUTHENTICATED state. Otherwise, the supplicant captures the EAP-Failure and transits to the HELD state.

4.2.2 EAP-MD5

The EAP-MD5 [5] is one of the most popular EAP types because it is easy to use. A user simply types in username and password to be authenticated. This is a simple and reasonable choice for wired LANs in which there is low risk for attackers to intercept the transmission. In wireless LANs, however, attackers can easily sniff a station's identity and password hash. Therefore, MD5 is more vulnerable than other authentication methods.

The MD5 algorithm described in [5] is implemented in md5.h and md5.cpp. We ported the MD5 algorithm from Open1x and then implemented a Graphical User Interface (GUI) for MS Windows. The GUI, shown in Fig. 7, consists of three input objects. Users only need to type in their usernames, passwords, and select the proper network interface to be authenticated.

4.2.3 EAP-TLS

The EAP-TLS [6] is based on TLS [21] to provide protected cipher-suite negotiation, mutual authentication, and key management. After the EAP-TLS negotiation is completed, the two end-points can securely communicate within encrypted TLS tunnel. Because TLS provides a way to use certificates for both user and server to authenticate each other, a user, in addition to being authenticated, can also authenticate the network. Therefore, forged APs could be detected. The supplicant and authentication server need to have valid certificates when using EAP-TLS. This, however, makes EAP-TLS difficult to manage in some extends.

Fig. 8 illustrates the authentication process and message exchanges of EAP-TLS in a WLAN [22]. After the authenticator receives the supplicant's identity in EAP-Response (flow 3), it initiates a RADIUS-Access-Request, which also carries the supplicant's identity. The RADIUS server then provides its certificate to the

supplicant and requests the supplicant's certificate. The supplicant validates the server's certificate and responds an EAP-Response which contains the supplicant's certificate. The supplicant also initiates the negotiation for cryptographic material. After the supplicant's certificate is validated, the server responds the cryptographic material for the session. The session keys derived in both ends could be used for data encryption.

The EAP-TLS is implemented in tls_funcs.h and tls_funcs.cpp. The GUI of EAP-TLS is shown in Fig. 9 which consists of three major input objects. The *select* button on the upper-right corner of the GUI is responsible for choosing the certificate in the supplicant. We use several functions from Windows SDK (Software Development Toolkit) in mfcDlg.cpp as below:

1. The CertOpenSystemStore() for opening a system certificate store.
2. The CryptUIDlgSelectCertificateFromStore() for selecting a new certificate using UI.
3. The CertGetNameString() for finding and printing the name of the subject of the certificate just retrieved.
4. The CertOpenStore() for opening the certificate store to be searched.
5. CertCloseStore() for closing the system certificate store.

To use these functions, the system header files of wincrypt.h and cryptuiapi.h must be included. Additionally, the libraries of crypt32.lib, advapi32.lib, and cryptui.lib must be linked. While we use these functions to get the certificate structure, we must replace SSL_CTX_use_certificate_file() by SSL_CTX_use_certificate_ASN1() in eapcrypt.cpp to receive the structure provided by Windows SDK functions.

The second object is the password for the selected certificate. Before using the supplicant's certificate, the user must type in the correct password for the certificate. The third object is to select the proper network interface to be authenticated. This is same as that in EAP-MD5.

4.2.4 EAP-TTLS

The EAP-TTLS [7] extends EAP-TLS to exchange additional information between client and server by using the secure tunnel established by TLS negotiation. A EAP-TTLS negotiation comprises two phases: the *TLS handshake phase* and the *TLS tunnel phase*. During phase one, TLS is used for client to authenticate server. Optionally, the server can also authenticate the client. Same as that in EAP-TLS, the authentication is done by using certificates. A secure TLS tunnel is also established after the phase-one handshake. In phase two, the secure TLS tunnel can be used for other information exchanges, such as additional user authentication key, communication of accounting information, etc.

In a WLAN environment, the EAP-TTLS usually is used as follows. First, TLS is used for a supplicant to authenticate the authentication server by using certificate. Once the authentication server is authenticated,

the authentication server authenticates the supplicant by the supplicant's username and password. The username and password are carried in the attribute-value pairs (AVPs) defined by the AAA server, which usually is a RADIUS server or Diameter server. The message exchanges are protected by the TLS tunnel established in phase one. The authentication of supplicant in phase two can use any non-EAP protocols such as PPP Authentication Protocols (PAP) [23], PPP Challenge Handshake Authentication Protocol (CHAP) [24], Microsoft PPP CHAP Extensions (MS-CHAP) [25], or Microsoft PPP CHAP Extensions, Version 2 (MS-CHAP-V2) [26]. Because only the authentication server needs to have a valid certificate, EAP-TTLS is more manageable than EAP-TLS.

The EAP-TTLS in WIRE1x is implemented in tls_funcs.h, tls_funcs.cpp, ttlsphase2.h, and ttlsphase2.cpp. The GUI of EAP-TTLS shown in Fig. 10 consists of five major input objects. The unprotected ID is used to represent the supplicant's identify for phase-one handshake. The username and password are used for phase-two authentication. The user also needs to choose one of the authentication protocols which can be PAP, CHAP, MS-CHAP, or MS-CHAP-V2 for phase-two authentication. Similarly, the user also needs to select the proper network interface to be authenticated.

4.2.5 PEAP

The PEAP [8] provides an encrypted and authenticated tunnel based on TLS. The EAP messages encapsulated inside the TLS tunnel, therefore, are protected against various attacks. Similar to EAP-TTLS, PEAP also comprises two phases. In first phase, a TLS session is negotiated. The client also authenticates the server by using certificate. Optionally, the server can also authenticate the client. In second phase, EAP messages are encrypted by using the key negotiated in phase one. The basic idea of PEAP and EAP-TTLS are identical. However, PEAP can only use EAP protocols in second phase, while EAP-TTLS can use EAP or non-EAP protocols.

When using PEAP in WLANs, typically an authentication server is authenticated by a supplicant based on the server certificate. A secure TLS tunnel is also created. A supplicant is then authenticated by using username and password, which are protected by the TLS tunnel.

The PEAP in WIRE1x is implemented in tls_funcs.h, tls_funcs.cpp, eapmschapv2.h, eapmschapv2.cpp, peapphase2.h, and peapphase2.cpp. The GUI of PEAP shown in Fig. 11 consists of three major input objects. A user must type in username and password for phase-two authentication. Currently, only EAP-MS-CHAP-V2 [27][‡] is supported. A user also needs to select the proper network interface to be authenticated.

To conclude Section 4.2, we provide comparison of the authentication mechanisms presented in this section by Table 4. Table 5 further compares the TLS-based protocols.

[‡]EAP-MS-CHAP-V2 encapsulates the MS-CHAP-V2 within EAP.

4.3 Open-Source Libraries

This section describes the open-source libraries used in WIRE1x. *WinPcap* and *Libnet* are used to capture/write packets from/to data link layer. *OpenSSL* is used for TLS-based authentication methods.

4.3.1 WinPcap

The WinPcap [10] is used for packet capture and network analysis in Win32 platform. It includes a kernel-level packet filter, a low-level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll which is based on libpcap version 0.6.2). It is in charge of the following tasks:

1. The pcap_findalldevs() in wire1xDlg.cpp prints the list of network interfaces. Therefore a user can select a proper network interface to be authenticated.
2. The pcap_dispatch() in os_generic.cpp captures packets from AP.
3. The setup_pcap() in os_generic.cpp can be used to adjust parameters in filter. Filter can make the supplicant receive EAP frames only.
4. The pcap_close() in os_generic.cpp shuts down WinPcap.
5. The pcap_open_live() in os_generic.cpp selects promiscuous mode or non-promiscuous mode for the network interface.

4.3.2 Libnet

The Libnet [11] is a generic networking API that provides access to several protocols. It is used only for libnet_write_link() in os_generic.cpp to write packets to AP.

4.3.3 OpenSSL

The OpenSSL [12] is an open-source toolkit which implements the Secure Sockets Layer (SSL) v2/v3 and TLS v1 protocols. It also includes a full-strength general purpose cryptography library. The TLS-based authentication methods in WIRE1x use OpenSSL library in eapcrypt.cpp. There are many OpenSSL functions in eapcrypt.cpp. Here, we only itemizes some of them:

1. The SSL_CTX_load_verify_locations() loads the server certificate in Privacy-Enhanced Electronic Mail (PEM) [28] format. As discussed earlier, all TLS-based authentication methods need server certificate in order to authenticate the server.
2. The SSL_CTX_use_certificate_file() loads the client certificate in Distinguished Encoding Rules (DER) [29] format. Alternatively, the SSL_CTX_use_certificate_ASN1() loads the client certificate in Basic Encoding Rules (BER) [29] format.

3. The SSL_CTX_use_PrivateKey_file() loads the client private key in PEM format. This function is used by EAP-TLS only.

4.4 GUI

The GUI is developed by MFC program of MS Visual C++. There are five GUI programs in the WIRE1x project. One of them is for the main program as shown in Fig. 12, which is responsible for selecting an EAP authentication method. The other GUIs are for the programs of md5.exe, tls.exe, ttls.exe, and peap.exe, which have been shown earlier. We also use thread function in mfcDlg.cpp to avoid blocking when GUI is executing. Please refer to the source code for details.

5 Real-World Applications

The WIRE1x has been practically used at the National Tsing Hua University (NTHU). At NTHU, each department/institute is responsible for the deployment of networking facilities in its own building(s). The university has no authority over the areas owned by the department/institute. The Computer & Communication Center (CCC) operated by the university is responsible for the networking facilities in public areas on campus. Thus, following standards is essential for roaming and integration of WLAN environments even inside the same university.

Both CCC and the Computer Science (CS) department at NTHU have deployed WLANs by using 802.1x and RADIUS to authenticate users. To roam between different administrative domains, both of their RADIUS servers could be connected together as shown in Fig 13. Assuming a user *abc* has an account *abc@nthu.edu.tw* owned by the CCC. Once the user roams into the WLANs covered by the CS department, the CS RADIUS server can authenticate the user by relaying the authentication messages back to the CCC RADIUS. The CS RADIUS server acts as a proxy client to the CCC RADIUS server. With only one account at CCC, the user still could roam into other WLANs.

6 Summary

This paper presents the motivation and contribution of WIRE1x. The implementation are discussed in details. By reading this paper, one should be able to examine the source code of WIRE1x in addition to using it. Currently, WIRE1x supports several wireless cards and provides various authentication methods, including EAP-MD5, EAP-TLS, EAP-TTLS, and PEAP. It is versatile comparing to many other implementations of IEEE 802.1x supplicant. WIRE1x has been downloaded worldwide. We hope WIRE1x will not only be used at NTHU, but will also play a crucial role for WLAN security and the interoperability of WLANs among different universities, companies, and organizations.

Acknowledgments

We thank other members of the WIRE Lab, especially Chin-Hsing Lin, Wen-Ting Wu, and Jui-Hung Yeh, for helping the development of WIRE1x. We also thank the support of the Computer and Communication Center, National Tsing Hua University, and the Computer Center, Department of Computer Science, National Tsing Hua University.

References

[1] IEEE Std 802.11i, "Part 11: wireless LAN medium access control (MAC) and physical layer (PHY) specifications. Amendment 6: medium access control (MAC) security enhancements," July 2004.

[2] J.-C. Chen, M.-C. Jiang, and Y.-W. Liu, "Wireless LAN security and IEEE 802.11i," *IEEE Wireless Communications (Accepted and to appear, 2004)*.

[3] IEEE Std 802.1X-2001, "IEEE standard for local and metropolitan area networks, port-based network access control," Oct. 2001.

[4] "Open1x." http://www.open1x.org/.

[5] R. Rivest, "The MD5 message-digest algorithm." IETF RFC 1321, Apr. 1992.

[6] B. Aboba and D. Simon, "PPP EAP TLS authentication protocol." IETF RFC 2716, Oct. 1999.

[7] P. Funk and S. Blake-Wilson, "EAP tunneled TLS authentication protocol (EAP-TTLS)." IETF Internet Draft, <draft-ietf-pppext-eap-ttls-05.txt>, work in progress, July 2004.

[8] A. Palekar, D. Simon, J. Salowey, H. Zhou, G. Zorn, and S. Josefsson, "Protected EAP protocol (PEAP), version 2." IETF Internet Draft, <draft-josefsson-pppext-eap-tls-eap-10.txt>, work in progress, Oct. 2004.

[9] "freeRADIUS." http://www.freeradius.org/.

[10] "WinPcap." http://winpcap.polito.it/.

[11] "Libnet." http://libnet.sourceforge.net/.

[12] "OpenSSL." http://www.openssl.org/.

[13] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)." IETF RFC 2865, June 2000.

[14] L. Blunk and J. Vollbrecht, "PPP extensible authentication protocol (EAP)." IETF RFC 2284, Mar. 1998.

[15] FIPS 197, "Advanced encryption standard (AES)." National Institute of Standards and Technology (NIST), Nov. 2001.

[16] D. Whiting, R. Housley, and N. Ferguson, "Counter with CBC-MAC (CCM)." IETF RFC 3610, Sept. 2003.

[17] "Lightweight extensible authentication protocol - LEAP." http://www.cisco.com/.

[18] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko, "Diameter base protocol." IETF RFC 3588, Sept. 2003.

[19] "wEAP." http://weap.sourceforge.net/.

[20] J. Kohl and C. Neuman, "The Kerberos network authentication service (v5)." IETF RFC 1510, Sept. 1993.

[21] T. Dierks and C. Allen, "The TLS protocol, version 1.0." IETF RFC 2246, Jan. 1999.

[22] Cisco Systems, "White paper: EAP-TLS deployment guide for wireless LAN networks." http://www.cisco.com/, 2004.

[23] B. Lloyd and W. Simpson, "PPP authentication protocols." IETF RFC 1334, Oct. 1992.

[24] W. Simpson, "PPP challenge handshake authentication protocol (CHAP)." IETF RFC 1994, Aug. 1996.

[25] G. Zorn and S. Cobb, "Microsoft PPP CHAP extensions." IETF RFC 2433, Oct. 1998.

[26] G. Zorn, "Microsoft PPP CHAP extensions, version 2." IETF RFC 2759, Jan. 2000.

[27] V. Kamath and A. Palekar, "Microsoft EAP CHAP extensions." IETF Internet Draft, <draft-kamath-pppext-eap-mschapv2-01.txt>, work in progress, Apr. 2004.

[28] B. Kaliski, "Privacy enhancement for Internet electronic mail: part iv: key certification and related services." IETF RFC 1424, Feb. 1993.

[29] ITU-T Rec. X.690, "ASN.1 encoding rules: specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER)," July 2002.

Table 1: Commercial 802.1x supplicants

Vendor	OS Supported	EAP Supported
Meetinghouse	MS Windows 98/ME/NT/2000/XP	MD5,LEAP,TLS,TTLS,PEAP
Funk	MS Windows 98/ME/2000/XP	MD5,LEAP
Microsoft	MS Windows 2000	TLS,PEAP
Microsoft	MS Windows XP	MD5,TLS
Microsoft	MS Windows XP with SP 1, SP 2	TLS,PEAP
Cisco	MS Windows/Mac OS/Linux	MD5,LEAP
secureW2	MS Windows 2000/XP/Pocket PC	TTLS

Table 2: Free 802.1x supplicants

Name	OS Supported	EAP Supported
Open1x	Linux/BSD/Mac OS	MD5,TLS,TTLS,PEAP,SIM
WIRE1x	MS Windows 98/2000/ME/XP (including SP1)	MD5,TLS,TTLS,PEAP

Table 3: Authentication servers

Vendor	EAP Supported
Funk Software	MD5,LEAP,TLS,TTLS
Interlink Networks	MD5,LEAP
Hewlett-Packard	MD5,LEAP
Microsoft	MD5,TLS,PEAP
Cisco Systems	MD5,LEAP,TLS
freeRADIUS	MD5,TLS,TTLS,PEAP,LEAP

Table 4: Comparison of authentication mechanisms

	EAP-MD5	EAP-TLS	EAP-TTLS	PEAP
Server Authentication	No	Public key (certificate)	Public key (certificate)	Public key (certificate)
Supplicant Authentication	Password hash	Public key (certificate or smart card)	Certificate, EAP, or non-EAP protocols	Certificate or EAP protocols
Mutual Authentication	No	Yes	Yes	Yes
Dynamic Key Delivery	No	Yes	Yes	Yes

Table 5: Comparison of TLS-based methods

	EAP-TLS (RFC 2716)	EAP-TTLS (Internet draft)	PEAP (Internet draft)
Basic Protocol Architecture	Establish TLS session and validate certificates for both client and server	(1) Establish TLS between client and TTLS server (2) Exchange attribute-value-pairs between client and server	(1) Establish TLS between client and PEAP server (2) Run EAP exchanges over TLS tunnel
Server Certificate	Required	Required	Required
Client Certificate	Required	Optional	Optional
Protection of User Identity	No	Yes, protected by TLS	Yes, protected by TLS

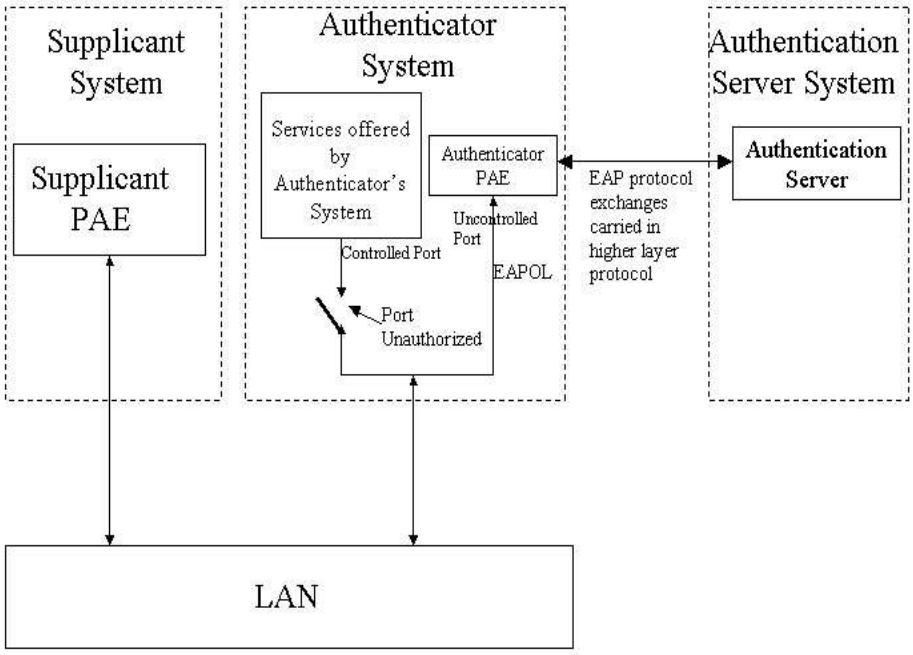

Figure 1: Supplicant, authenticator, and authentication server

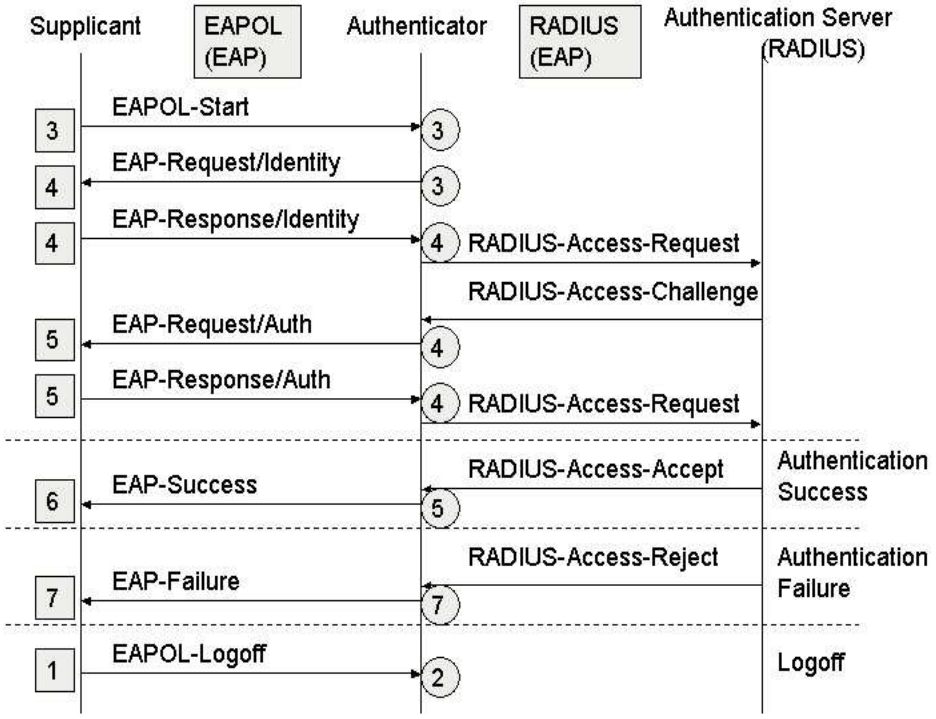

Figure 2: A typical 802.1x message flow

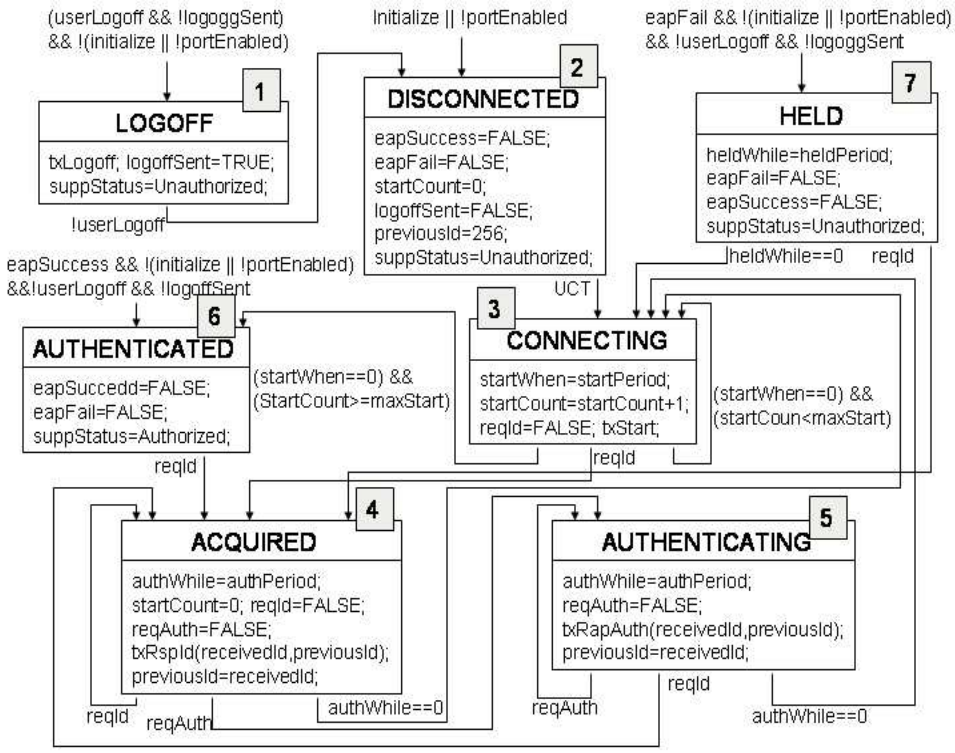

Figure 3: Supplicant PAE state machine

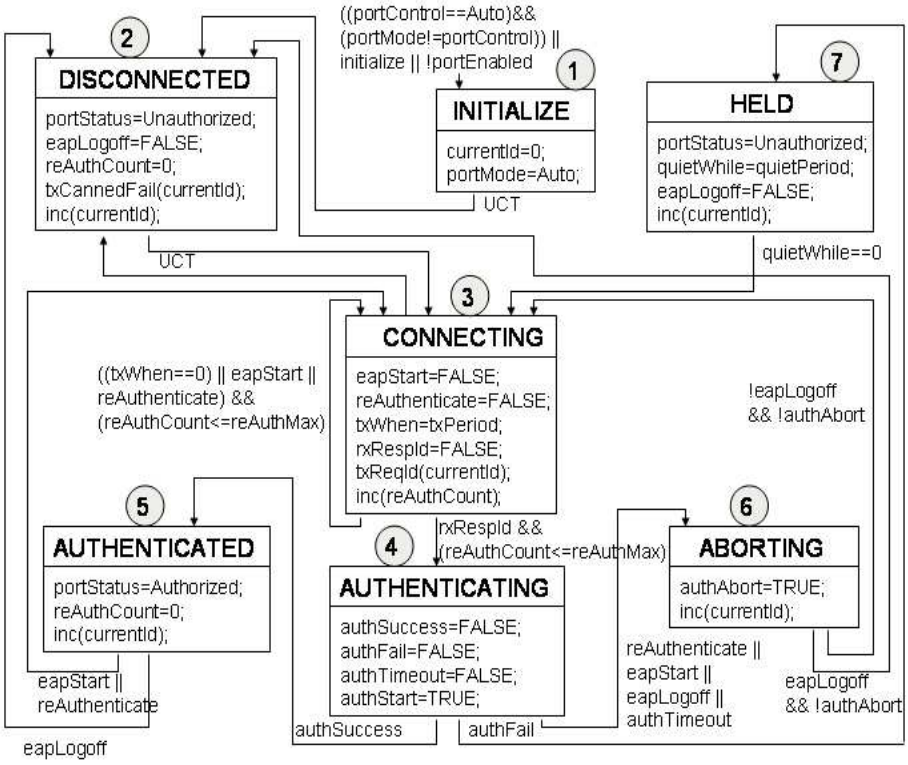

Figure 4: Authenticator PAE state machine

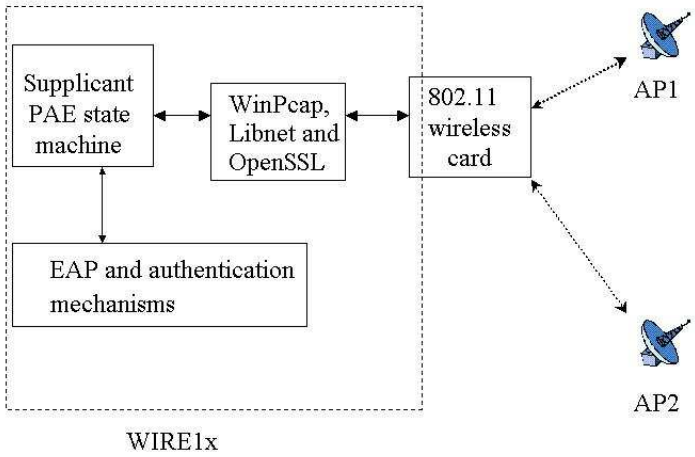

Figure 5: Software architecture of WIRE1x

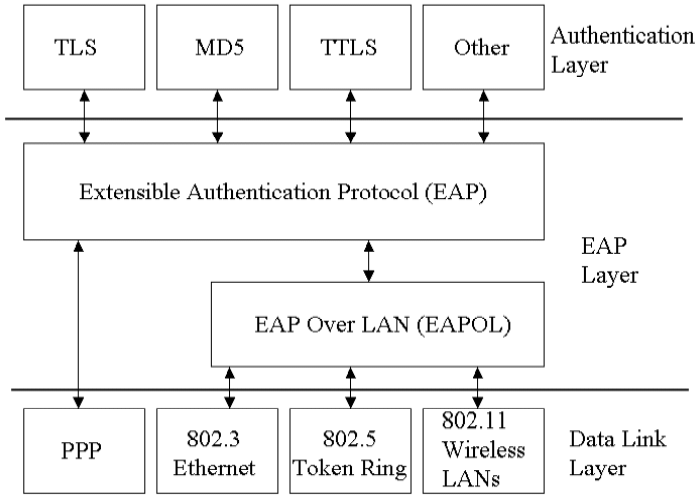

Figure 6: EAP and associated layers

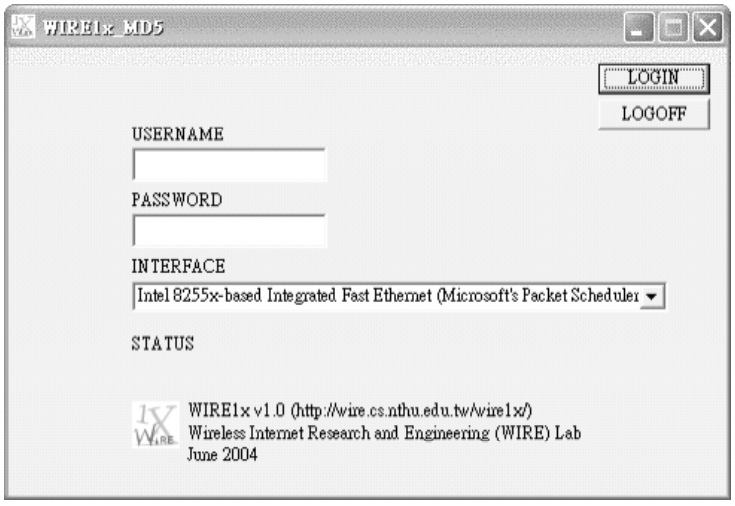

Figure 7: GUI of EAP-MD5

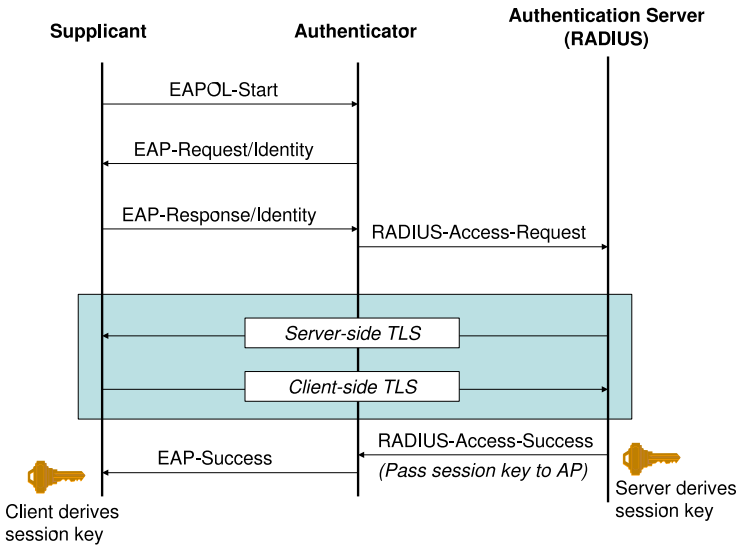

Figure 8: Message flow of EAP-TLS

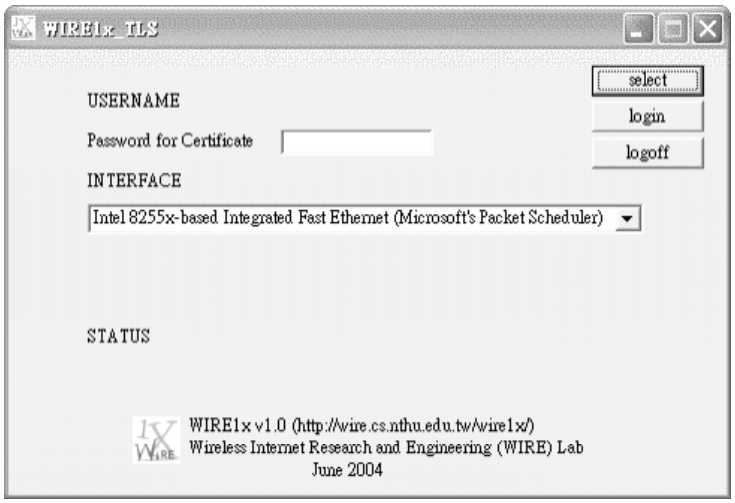

Figure 9: GUI of EAP-TLS

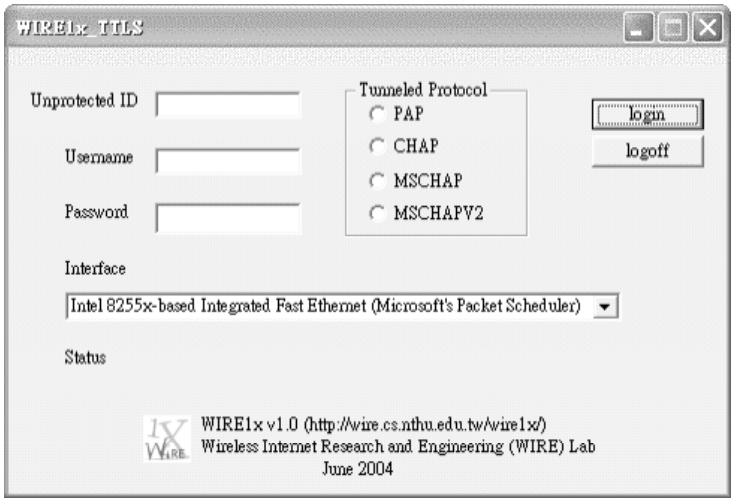

Figure 10: GUI of EAP-TTLS

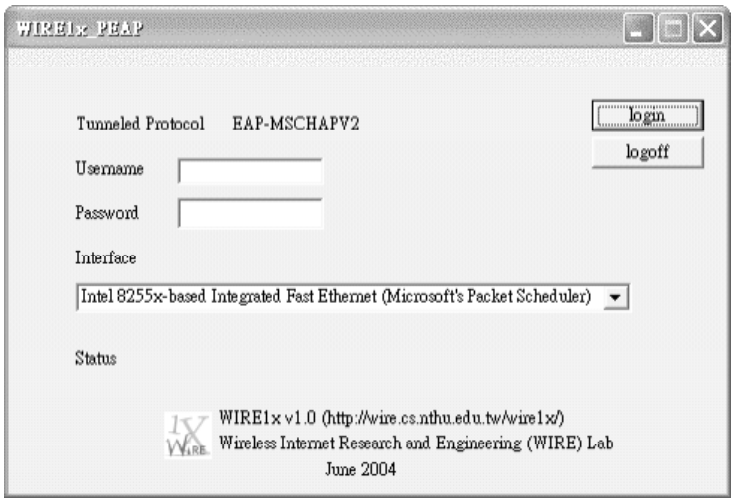

Figure 11: GUI of PEAP

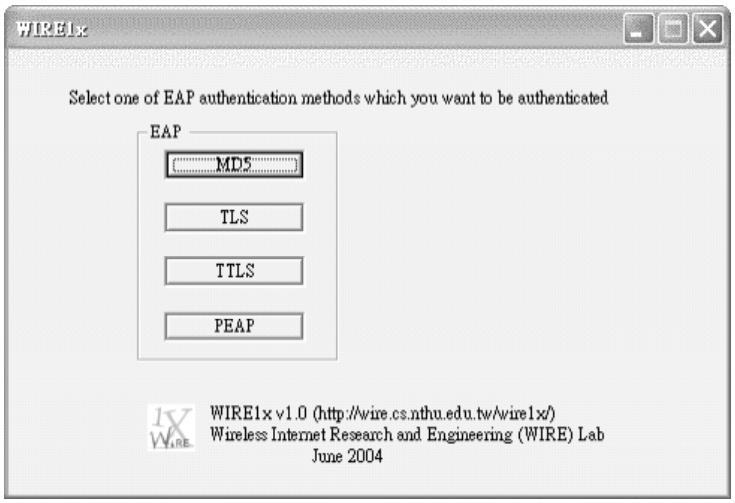

Figure 12: GUI of main program

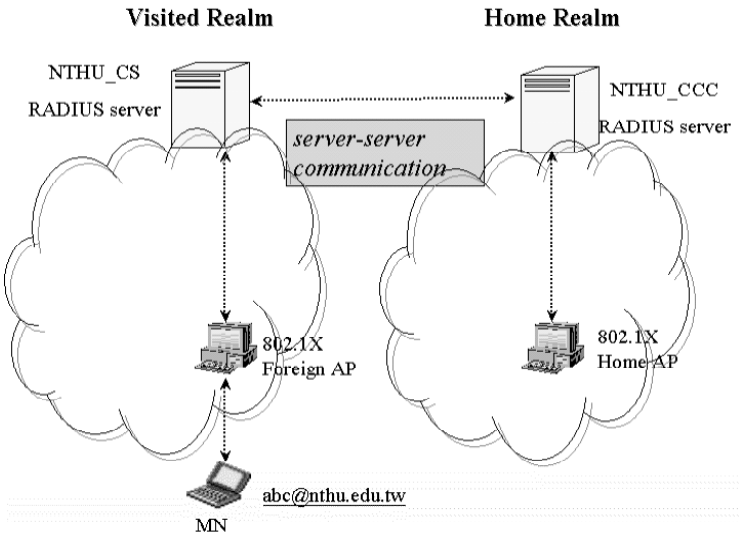

Figure 13: Authentication on visited network