J. Indian Inst. Sci., July–Aug. 2006, **86**, 389–396

Secure password-based authentication in WLAN

M. KESHAVA*
SASKEN Communication Technologies Ltd, Bangalore 560 071.
emails: keshava.gowda@gmail.com; keshava_gowda@rediffmail.com
Mobile: +91-0-9448573494

Received on May 12, 2005; Revised on April 9, 2006 and July 3, 2006

Abstract

The EAP-TLS is a de-facto authentication protocol in 802.11i system. This protocol provides digital certificate-based mutual authentication. The protocol performs secure password-based client/supplicant authentication instead of certificate-based authentication. This paper illustrates the modifications on EAP-TLS protocol to achieve secure password-based user/client authentication, achieving the goal of EAP-TTLS without forming a logical tunnel between a supplicant and authentication server. A comparison between the proposed technology and EAP-TTLS brings out the performance enhancements possible with this technology. The proposed system supports an optional mutual password-based authentication during session resumption.

Keywords: EAP, TLS, TTLS, AVP, PRF, MD5, SHA, PMK, MAC, RADIUS.

1. Introduction

The EAP-TLS [1] (extensible authentication protocol-transport layer security) and EAP-TTLS (EAP-tunneled TLS) [2] are two widely used mutual authentication protocols in WLAN environment. The EAP-TTLS protocol defines two phases for mutual authentication. The first phase, i.e. the handshake phase corresponds to the execution of the EAP-TLS protocol to perform the digital certificate-based server authentication and to create a secure logical tunnel for password-based user/client authentication. The second phase of the EAP-TTLS protocol, i.e. the tunnel phase, involves the execution of a pre-configured password-based authentication protocol. In this context, a secure logical tunnel between a client/supplicant [3] and AS (authentication server) [3] means all messages between them are protected using TLS [4] keys generated at the end of phase one. Due to this tunnel creation, the EAP-TTLS protocol hides the password-based user/client authentication messages from intruder.

This paper discusses the technique of performing password-based authentication during EAP-TLS handshaking, thus eliminating the need for the tunnel phase (phase 2) of EAP-TTLS and hence condensing the authentication conversation. The modifications done to the EAP-TLS are as follows.

(i) A new handshake message named *password_sign_request* is added to the TLS protocol.

*Present address: Texas Instruments (India) Pvt Ltd, Bagmane Tech Park, 66/3, Byrasandra, Adjacent to LRDE, C. V. Raman Nagar P.O., Bangalore 560 093. e-mail: keshava_mgowda@ti.com

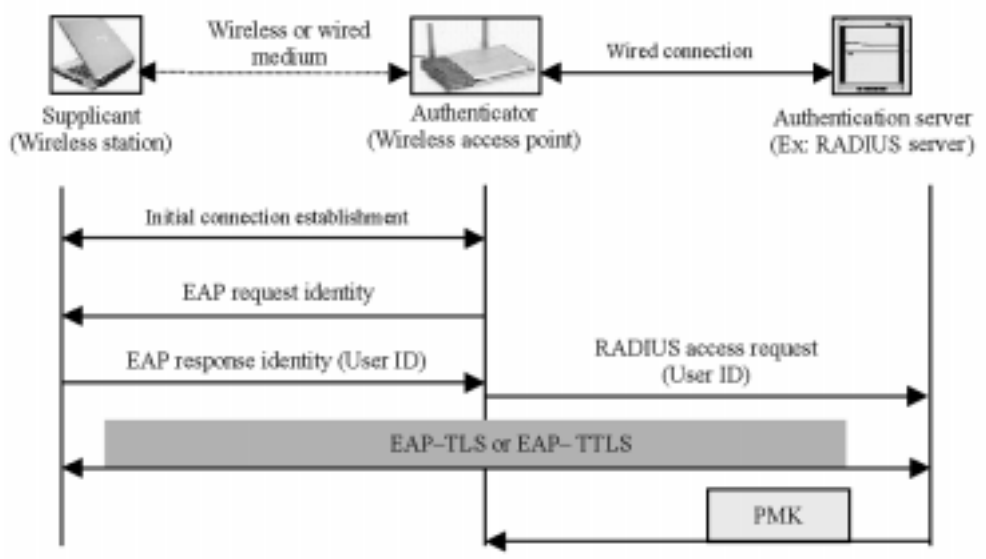

FIG. 1. EAP-based protocols.

(ii) The finished_label [4, section 7.4.9-finished message] is modified to determine the password-based signature on verify_data [4] of TLS finished message.
(iii) The sequence of flow of EAP-TLS messages is modified to perform mutual password-based authentication in case of session resumption.

The proposed protocol follows EAP-TLS protocol for PMK (pairwise master key) [5] generation, retry behavior, message fragmentation and reassemble process. In the proposed system, both AS and client/supplicant system share a common password string for each user/client. Usually, the AS stores the multiple user_ids/peer_ids and corresponding shared passwords in a database or on a file system. The AS extracts the shared password depending on the unique user_id/peer_id received from authenticator. Figure 1 abstracts the EAP [6] conversation. Note that the client/supplicant system sends the user_id/peer_id in the EAP response Identity message to the authenticator. The AS receives this user-id from the authenticator using RADIUS (remote authentication dial in user service) [7] protocol.

2. Related work

The EAP-PEAP v2 [8] (EAP-protected EAP, version 2) and EAP-FAST [9] (EAP-flexible authentication via secure tunneling) protocols also perform the secure password-based client authentication using EAP-TLS for tunnel creation. The EAP-PEAP v2 protocol allows only EAP-based protocols like EAP-MD5 (message digest 5), EAP-MSCHAP (MicroSoft challenge handshake authentication protocol) inside EAP-TLS tunnel, whereas EAP-TTLS allows any password-based authentication inside the EAP-TLS tunnel, and is not limited to EAP-based protocols. The EAP-FAST protocol is the successor of LEAPv2 (light-weighted EAP protocol, version 2). This protocol provides the flexibility to create a tunnel with a pre-shared key. This key is also called PAC (protected access credential) key.

3. Password_sign_request—A new handshake message to TLS protocol

This paper follows the same presentation language of TLS protocol for describing the format of *password_sign_request* message. The inclusion of *password_sign_request* message to the handshake message structure of the TLS protocol is shown below.

```
struct {
        HandshakeType msg_type;
        uint24     length;
        select (HandshakeType) {
                case password_sign_request:
                        PasswordSignRequest;
        } body
} Handshake;
```

```
struct {
        /* No data */
} PasswordSignRequest;
```

The *password_sign_request* message does not contain a body. The handshake type number of this message notifies the receiver that it has to determine the password-based signature on verify_data. This verify_data forms the body of final outgoing TLS finished handshake message. The verify_data with password-based signature authenticates the sender of TLS finished message. Section 4 describes the verify_data calculation. The implementation of this paper assigns decimal number 18 as a handshake type number of the *password_sign_request* message.

4. Secure password-based signature

In TLS protocol, the server uses the string 'server finished' and client uses the string 'client finished' as finished_label. This finished_label is an input to PRF (pseudo-random function) [4]. The verify_data is an output of PRF. This verify_data contains the hash of MD5 and SHA1 (secure hash algorithm, version 1) digests of TLS handshake messages. By concatenating the shared password and finished_label, the server and supplicant can obtain the verify_data with password-based signature. If the client/supplicant has received the *password_sign_request* message then it should use the finished_label {shared_password + 'client finished'} ('+' *denotes concatenation*) to generate verify_data. Server should use the same finished_label to verify the supplicant's signature on the received verify_data. In case of session resumption, if the server receives the *password_sign_request* handshake message then it uses the finished_label {shared_password + 'server finished'}. Client/supplicant should use the same label to do the password-based authentication of server. Since the TLS finished message is always protected with negotiated cipher suite [4] and TLS keys, the verify_data is secure from dictionary attacks and reply attacks. This system assumes that the shared password is securely loaded (or it could be manually loaded) to both the supplicant and the AS along with the user details. Shared password is never transmitted over the network during authentication.

5. Creation of new session

While creating a new session, the server sends *password_sign_request* message before server hello done message (Fig. 2). As a response to this message, the client/supplicant

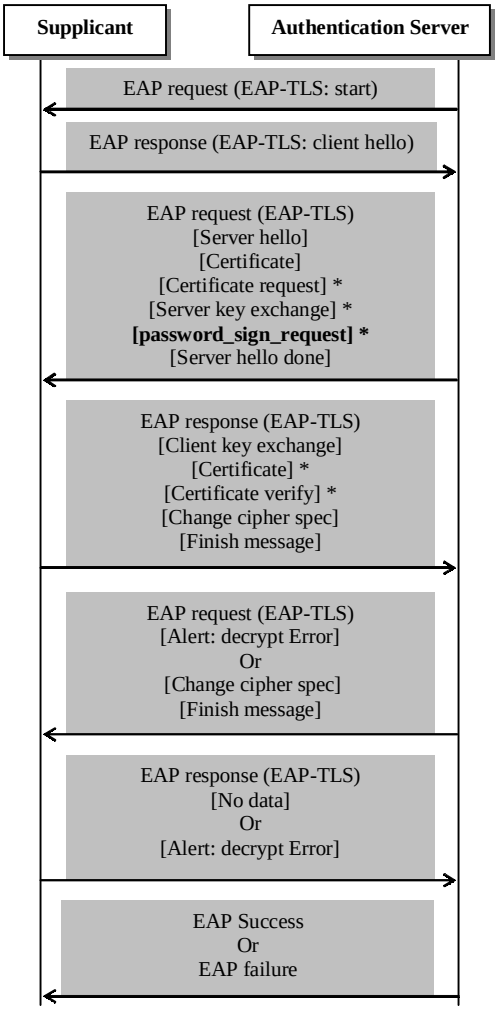

FIG. 2. New session creation.

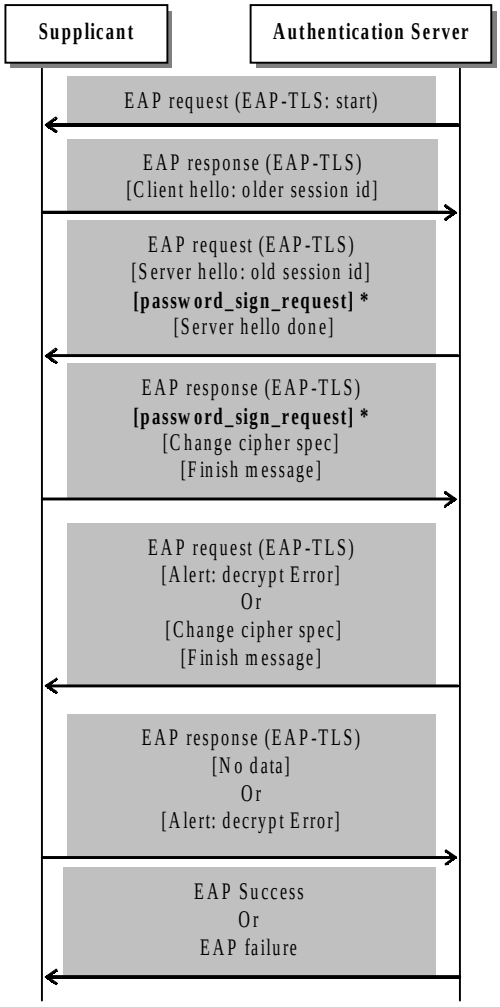

FIG. 3. Session resumption.

Note: In Figs 2 and 3, optional messages are shown with * symbol and an additional handshake message *password_sign_request* in shown in bold.

should use a new finished_label, as described in Section 4 to generate verify_data of the finish handshake message. The AS/server decrypts the finished message and verifies the password-based signature on the verify_data of the TLS finish handshake message. If the verification fails then it sends EAP-TLS alert message 'decrypt error' to supplicant; otherwise, change cipher spec and finished messages are sent to the supplicant. The client/supplicant decrypts and verifies the received TLS finished message as described in TLS protocol. If the verification fails, then it sends EAP-TLS alert message 'decrypt error' to server; otherwise, an acknowledgment EAP-TLS packet will be sent to AS. Finally, as a response, AS sends either EAP-success or EAP-failure packet to authenticator, which forwards the received EAP packet to supplicant.

6. Session resumption

While resuming an older session, the server can send the *password_sign_request* message after the server hello message (Fig. 3). When the client/supplicant receives this *password_sign_request* message, it should use the new finished_label, as defined in Section 4, to generate the verify_data of finish handshake message. The client/supplicant can also send this message after receiving the server hello done message. When the server receives this message then it should use the new finished_label, as defined in Section 4, to calculate the verify_data of the finish handshake message. If the client or server fails to verify TLS finished message then it should send the EAP-TLS alert message 'decrypt error' to sender. In EAP-TLS protocol, while resuming an older session, the AS/server transfers change cipher spec and TLS finish message along with server hello message. It does not transmit the server hello done message. But, in the proposed protocol**,** to provide an optional password-based mutual authentication during session resumption, the AS/server transfers the sever hello done message following an optional *password_sign_request* message. The AS/server then sends the change cipher spec and finish message after successful verification of finish message sent by the supplicant/client. The client sends an EAP-acknowledgement packet if it successfully verifies the TLS finished message.

7. Implementation and experimental results

This system can be implemented by modifying the code of the following software modules.

(a) SSL and TLS implementations of OpenSSL, version: 0.9.7c
(b) X-supplicant, version: 1.2.2
(c) Free RADIUS server, version: 1.0.5

These are free software modules available for the Linux operating system. The OpenSSL package maintains a finite state machine for parsing and processing TLS messages. The implementation of this system handles the *password_sign_request* message by adding an additional state to the finite state machines of TLS client and TLS server. The test environment consists of two PCs. A wireless NIC (network interface card) is attached to one PC which acts as wireless client/supplicant and it executes X-supplicant software module. Another PC executes the RADIUS server which has the implementation of modified EAP–TLS server. The RADIUS server is attached to Cisco Aironet 1130 AG Access Point via conventional 100 mbps ethernet. In the X-supplicant and RADIUS server, the maximum EAP–TLS packet size is chosen as a default 1024 bytes. In this test setup, the sender (either server or client) transmits only one certificate. This certificate contains 1024 bit RSA public key and MD5 signature. For measuring EAP–TTLS performance, the EAP-MD5 protocol is used as a tunneled password-based user/client authentication protocol and the size of the password is 16 bytes. The modified EAP–TLS protocol also uses a 16-byte password string. The cipher suite, selected between AS and client/supplicant, affects the performance of EAP–TLS, EAP–TTLS and Modified EAP–TLS protocol, i.e. the implementation of this paper. Table I shows the performance numbers of these authentication protocols. These are measured at the X-supplicant. The time stamps are recorded when the client/supplicant sends the client hello message and when it receives the EAP-success message. The difference between these time stamps is measured as time taken by an EAP-based protocol.

Table I
Performance numbers of EAP–TLS, EAP–TTLS and modified EAP–TLS

#	Selected cipher suite	Authentication protocol	Time (seconds)	
			New session creation	Session resumption
A	TLS_RSA_WITH_RC4_128_SHA	EAP–TLS	0.05881	0.02351
		EAP–TTLS	0.06685	0.02397
		Modified EAP–TLS	0.04794	0.02705
B	TLS_RSA_WITH_DES_CBC_SHA	EAP–TLS	0.06041	0.02357
		EAP–TTLS	0.06806	0.02418
		Modified EAP–TLS	0.04817	0.02715
C	TLS_RSA_WITH_RC4_MD5	EAP–TLS	0.05879	0.02348
		EAP-TTLS	0.06680	0.02379
		Modified EAP–TLS	0.04787	0.02701
D	TLS_RSA_WITH_3DES_EDE_CBC_SHA	EAP–TLS	0.06147	0.02388
		EAP–TTLS	0.07105	0.02492
		Modified EAP–TLS	0.04710	0.02742

In EAP–TTLS, while creating a new session, all EAP–MD5 messages are prefixed with an AVP (attribute value pair) [2] header. This message containing EAP–MD5 packet is exchanged as an encrypted TLS message. The MAC (message authentication code) [4] is calculated on each TLS message. Every TLS message, concatenated with its MAC, is encrypted using TLS keys. This lowers the performance of the EAP–TTLS protocol when compared to both the EAP–TLS protocol and the proposed protocol, i.e. modified EAP–TLS. Figure 4(a) shows the performance difference between these protocols while creating a new session. Note that the modified EAP–TLS protocol is very fast when compared to EAP–TTLS and EAP–TLS, as there is no client certificate transmission as in EAP–TLS protocol and there are no tunneled EAP–MD5 messages as in EAP–TTLS. While resuming an older session, both EAP–TLS and EAP–TTLS protocols skip the authentication, whereas the modified EAP–TLS protocol achieves password-based mutual authentication with an additional *password_sign_request* message. This causes a small hit on the performance of the modified EAP-TLS protocol (Fig. 4(b)).

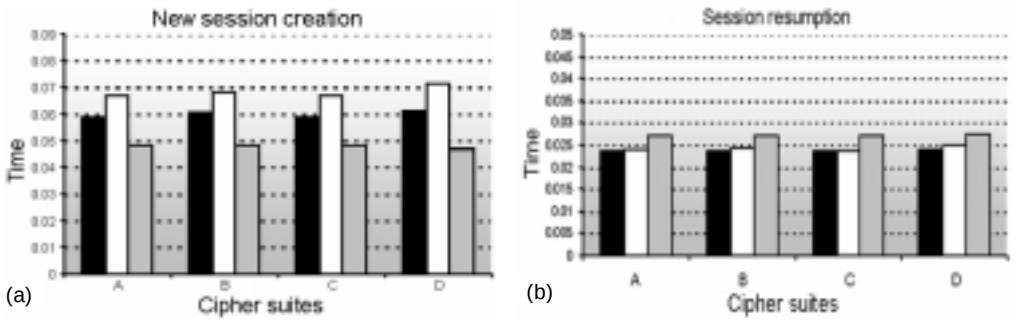

FIG. 4. Performance differences between EAP–TLS, EAP–TTLS and modified EAP–TLS while (a) creating new session and (b) during session resumption. ■ EAP–TLS; □ EAP–TTLS; ▩ Modified EAP–TLS.

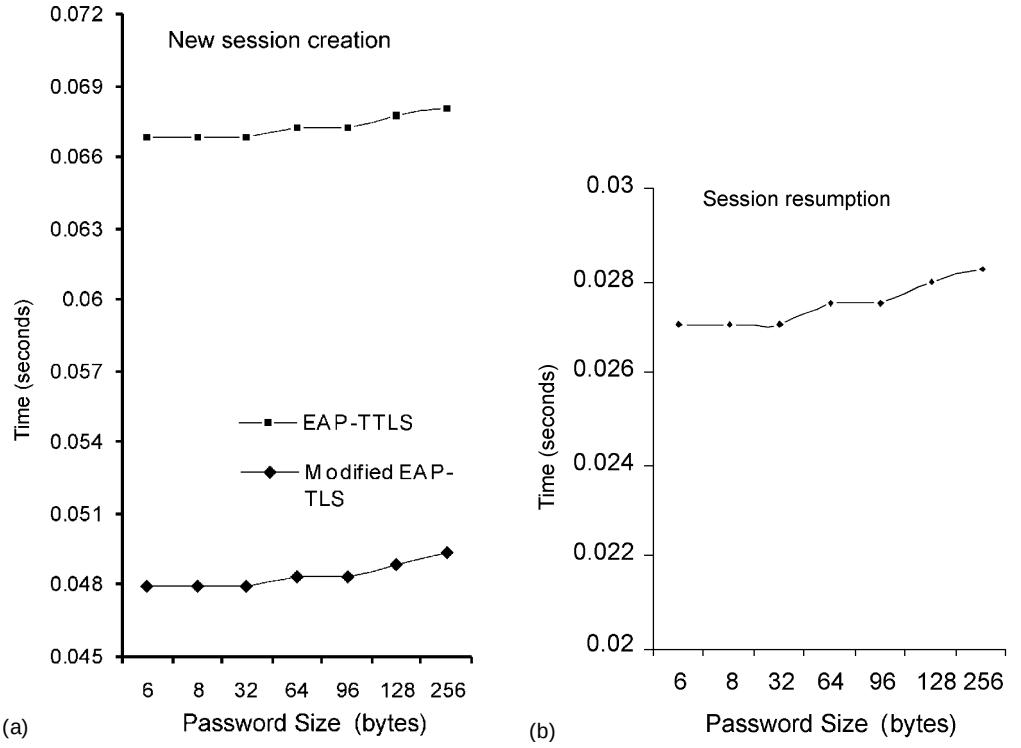

FIG. 5. Performance variations of (a) EAP–TTLS and modified EAP–TLS, and (b) modified EAP–TLS protocols with respect to the size of the password. Selected cipher suite is TLS_RSA_WITH_RC4_128_SHA.

Increasing the size of the password reduces the performance of EAP–TTLS and the modified EAP–TLS protocol. For the analysis of EAP–TTLS performance, the test setup uses EAP–MD5 which in turn uses the MD5 algorithm to calculate the digest for the challenge sent by the server. The modified EAP–TLS protocol uses the PRF to determine verify_data of the TLS finish message. Since the PRF internally uses both MD5 and SHA1 algorithms, the effect of increasing the size of the password is slightly more on the modified EAP–TLS protocol than on EAP–TTLS. But the overall performance of the proposed system is still much higher than the performance of EAP–TTLS protocol because the AVP layer is removed. The gradual reduction in the performance of these two protocols with increasing password size is shown in Fig. 5(a). While resuming an older session, only the modified EAP–TLS protocol suffers from performance thrashing as the password size increases. This behavior is depicted in Fig. 5(b).

8. Conclusion

The proposed system extends the EAP–TLS protocol with an additional handshake message, *password_sign_request*. According to Challenge-Response authentication model, the MD5 and SHA1 digests of the handshake messages can be viewed as a challenge and the verify_data of the finish handshake message can be called as a response signature element.

Following are the key differences between the authentication mechanism proposed in this paper and EAP–TTLS protocol.

(1) This modified EAP–TLS protocol eliminates the need for the tunnel phase (phase 2), which is required in the EAP–TTLS protocol. This reduces the number of messages required for mutual authentication while creating a new session and thus speeds up authentication. While the elimination of the AVP layer means that the deployment of password-based authentication protocols such as EAP–MD5, EAP–MSCHAP or any other vendor-specific protocol is no longer possible, the authentication is still secure because this protocol hides both the challenge and the password-based response signature.
(2) Both EAP–TLS and EAP–TTLS protocols skip authentication while resuming an older session. The modified EAP-TLS protocol provides flexibility of performing password-based mutual authentication in case of session resumption but this causes a slightly lower performance.

Acknowledgments

It is a pleasure to acknowledge Mr G. T. Raju Assistant Professor, Department of Computer Science, and Mr. Ashok Kumar, Assistant Professor, Department of Information Science, BMS College of Engineering, Bangalore, for the guidance provided throughout this research work. The Device Driver team members of Semiconductor Group of the SASKEN Communication Technologies and Mr Vinay Keelara, Member of Technical Staff, EDGE Dynamics, Philadelphia, USA, deserve special acknowledgement for enthusiastic support during the finalization of this paper. Last but not the least, this paper owes great deal of thanks to Mr K Sreenivasa Rao, Assistant Editor, *Journal of the Indian Institute of Science*, and to an anonymous reviewer for the encouragement and involvement in innumerable reviews to elevate the quality of this research.

References

1. B. Aboba, and D. Simon, *PPP EAP TLS authentication protocol*, RFC 2716, Microsoft (1999).
2. Paul Funk, and Simon Balke-Wilson, *EAP tunneled TLS protocol*, Draft-ietf-pppext-eap-ttls-01.txt (2002).
3. ANSI/IEEE STD 802.1X, *Standard for port based network access control* (2001).
4. T. Dierks, and C. Allen, *The TLS protocol version 1.0*, RFC2246 (1998).
5. ANSI/IEEE STD 802.11i, *Wireless LAN medium access control and physical layer* (*PHY*) *specifications: Medium access control (MAC) security enhancements* (2004).
6. L. Blunk, and J. Vollbrecht, *PPP extensible authentication protocol (EAP)*, RFC 2284 (1998).
7. C. Rigney, A. Rubens, W. Simpson, and S. Willens, *Remote authentication dial in user service* (*RADIUS*), RFC 2865 (2000).
8. Ashwin Palekar, Dan Simon, Glen Zorn, Joe Salowey, Hao Zhou, and S. Josefsson, *Protected EAP protocol version 2*, Draft-josefsson-pppext-eap-tls-eap-07.txt.
9. N. Cam-Winget, D. McGrew, J. Salowey, and H. Zhou, *The flexible authentication via secure tunneling extensible authentication protocol method (EAP-FAST)*, Draft-cam-winget-eap-fast-03.txt.
10. ANSI/IEEE STD 802.11, *Wireless LAN medium access control and physical layer (PHY) specifications* (1999).
11. http://www.openssl.org, *Open source SSL & TLS implementation*.
12. http://www.freeradius.org, *Open source free RADIUS server implementation*.
13. http://open1x.sourceforge.net, Open source 802.1x supplicant and authenticator implementation.