CHAPTER 7

EAP Authentication Protocols for WLANs

The second in the WLAN authentication trilogy of chapters, this chapter examines the various authentication protocols such as the Extensible Authentication Protocol (EAP), Protected EAP (PEAP), the Lightweight Extensible Authentication Protocol (LEAP), and EAP- Flexible Authentication via Secure Tunneling (EAP-FAST). This chapter begins with a look at the fundamental concepts and contexts of authentication and access control; next, it discusses the various protocols such as EAP and 802.1x.

Notice the slow progression out of the basic 802.11 standards as you begin to leverage other standards: IEEE, the Internet Engineering Task Force (IETF), and sometimes even proprietary standards. You will see how the various protocols add more security features such as encrypted tunnels for exchanging various information (authentication, credentials, and other data), dynamic key distribution and rotation, authenticating the user rather than the device, and applying identity-based mechanisms and systems that are part of the administrative domain in enterprises.

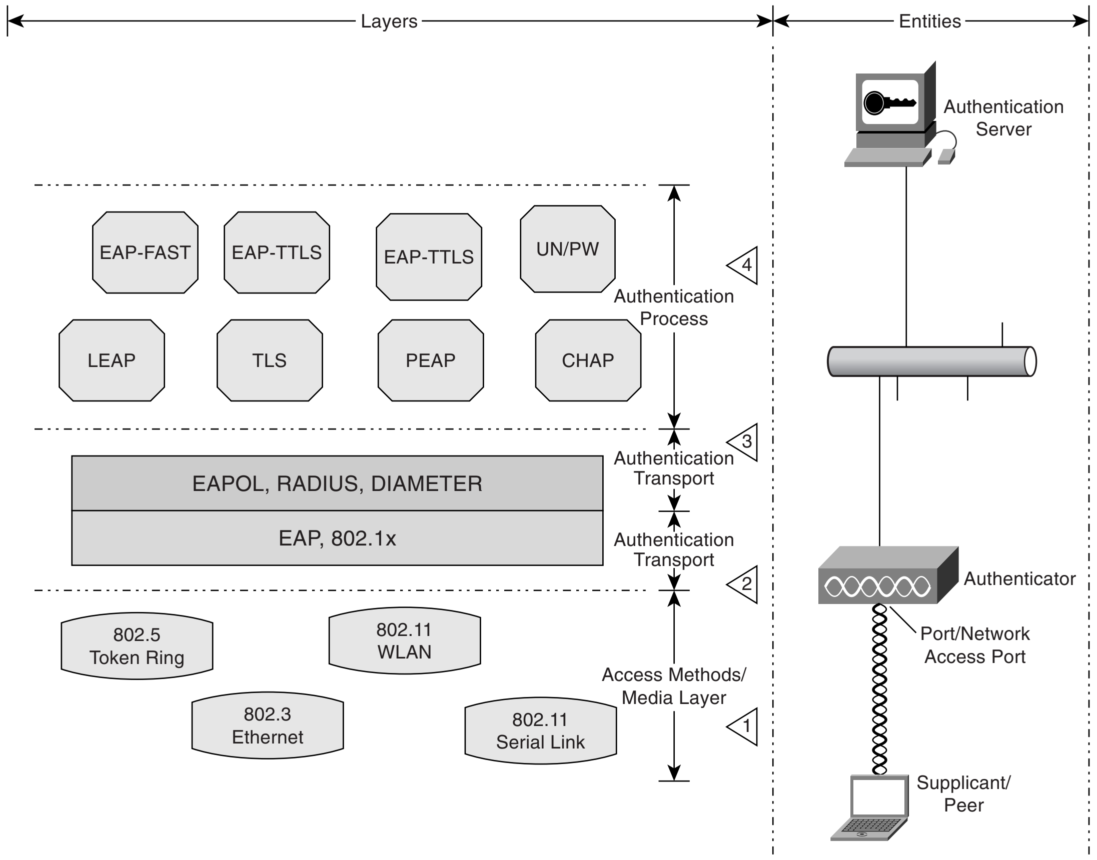
Access Control and Authentication Mechanisms

Before allowing entities to access a network and its associated resources, the general mechanism is to authenticate the entity (a device and/or user) and then allow authorization based on the identity. The most common access control is binary: It either allows access or denies access based on membership in a group.

NOTE Extending access control, especially to the wireless world, means a more finely grained authorization; for example, you can allow access to the network and its resources for internal employees and allow Internet access for guests. Employees are also working on federations, so access can be allowed based on the entity's membership in identity federations—for example, intercollege access to researchers, interorganization access based on collaboration on certain projects, and other similar groups and roles.

The different layers, standards, and conceptual entities in the EAP/802.1x world are seen in Figure 7-1.

Figure 7-1 *Layered Authentication Framework*



The Three-Party Model

The authentication is based on a three-party model: the supplicant, which requires access; the authenticator, which grants access; and the authentication server, which gives permission.

The supplicant has an identity and some credentials to prove that it is who it claims to be. The supplicant is connected to the network through an authenticator's port that is access controlled. The port concept is important because it acts as the choke point for the supplicant's access to the network resources. The access to the network can be controlled at a single point. The supplicant is called a *peer* in the IETF RFCs and drafts.

NOTE In the wireless world, the most common supplicant is the STA (Station) (laptop or PDA), and the authenticator is the access point (AP). The STA to AP cardinality is 1:1. (That is, one STA can, at one time, connect to the network through only one AP.) This restriction is tailor made for the EAP/802.1x concept of an access-controlled port.

The authenticator itself does not know whether an entity can be allowed access; that is the function of the authentication server. In the IETF world, the authenticator is referred to as the *network access server (NAS)* or Remote Address Dial-In User Service (RADIUS) client.

NOTE In many cases, the authenticator and the authentication server roles can be performed by one device, such as the 802.11 AP.

Let's look at the big picture before discussing the details. The supplicant initiates an access request, and the authenticator starts an EAP message exchange. (In the stricter sense of the standards, such as 802.1x, the supplicant does not necessarily *always* initiate the access request; the authenticator can initiate an authentication request when it senses a disabled-to-enabled state transition of a port.) At some point, the authenticator communicates with the authenticator server, which decides on an authentication protocol. A set of exchanges then occurs between the supplicant, the authenticator, and the server; at the end of this exchange, a success or failure state is reached. If the authentication succeeds, the authenticator allows network access to the supplicant through the port. The authenticator also keeps a security context with the supplicant-port pair. This context could trigger many things, including timeout if the authentication is only for a period of time (for example, the billed access in public WLAN scenario).

Layered Framework for Authentication

As shown in Figure 7-1, the authentication model is a layered one and has well-defined functionalities and protocols defining each layer and the interfaces between them. The access media (Step 1 in Figure 7-1) can be any of the 802 media: Ethernet, Token Ring, WLAN, or the original media in the serial Point-to-Point Protocol (PPP) link. The EAP specifications provide a framework for exchanging authentication information (Step 2 in Figure 7-1) after the link layer is established. The exchange does not even need IP. It is the function of the transport protocol layer (Step 3 in Figure 7-1) to specify how EAP messages can be exchanged over LAN, which is what 802.1x (and to some extent some parts of 802.11i) does. The actual authentication process (Step 4 in Figure 7-1) is the one that defines how and what credentials should be exchanged. Bear in mind that this framework still does not say how the authorization should be done, such as what decisions are made and when. This functionality is completely left to the domain.

Table 7-1 lists the major standards and efforts in the authentication framework domain. This chapter covers the different flavors of EAP. Hopefully, this table will enable you to dig deeper into the areas in which you are interested.

Table 7-1 *Specifications and Standards in the Authentication Framework Domain*

Mechanism	**Specification**	**Description**
Domain: Access Method		
PPP	RFC 1661: The Point-to-Point Protocol (PPP)	
802.3, 802.5, 802.11 and other standards	Various	IEEE access media standards
Transport Layer Security (TLS)	RFC 2246: Transport Layer Security Version 1.0	
	RFC 3268: AES Cipher Suit for TLS	
	RFC 3546: TLS extensions	
Domain: Authentication Exchange		
EAP	RFC 2284: PPP Extensible Authentication Protocol (EAP)	Original 1998 EAP standard
	RFC 3579: RADIUS Support for EAP	Was RFC 2284bis Will supersede RFC 2284
	draft-urien-eap-smartcard-03.txt	EAP-Support in SmartCard
	draft-funk-eap-md5-tunneled-00.txt	EAP MD5-tunneled authentication protocol
	draft-mancini-pppext-eap-ldap-00.txt	EAP-LDAP protocol
	draft-haverinen-pppext-eap-sim-12.txt	EAP SIM authentication
	draft-arkko-pppext-eap-aka-11.txt	EAP AKA authentication
	draft-tschofenig-eap-ikev2-02.txt	EAP IKEv2 method
	draft-salki-pppext-eap-gprs-01.txt	EAP GPRS protocol
	draft-aboba-pppext-key-problem-07.txt	EAP key management framework
	draft-jwalker-eap-archie-01.txt	EAP Archie protocol
	draft-ietf-eap-statemachine-01	State machines for EAP peer and authenticator

Table 7-1 *Specifications and Standards in the Authentication Framework Domain (Continued)*

Mechanism	Specification	Description
802.1x	IEEE Std. 802.1X-2001	Port-based network access control
	802.1aa	Revision of the 802.1x, work-in-progress
Domain: Authentication Process		
RADIUS	RFC 2865: RADIUS	Current RADIUS specification Supersedes RFC 2138, which in turn supersedes RFC 2058
	RFC 2866: RADIUS Accounting	Defines protocol for carrying accounting information between authenticator and authentication server Supersedes RFC 2139, which in turn supersedes RFC 2059
	RFC 2867: RADIUS Accounting Modifications for Tunnel Protocol Support	Updates RFC 2866
	RFC 2868: RADIUS Attributes for Tunnel Protocol Support	Updates RFC 2865
	RFC 2809: Implementation of L2TP Compulsory Tunneling via RADIUS	
	RFC 2869: RADIUS Extensions	Adds attributes for carrying AAA information between the authenticator (NAS) and authentication server (shared accounting server)
	RFC 3576: Dynamic Authorization Extensions to RADIUS	
	RFC 2548: Microsoft Vendor-Specific RADIUS Attributes	
	RFC 3575: IANA Considerations for RADIUS	Describes best practices for registering RADIUS packet types Updates Section 6 of RFC 2865
	RFC 3580: IEEE 802.1x Remote Authentication Dial-In User Service (RADIUS) Usage Guidelines	
	RFC 3162: RADIUS and IPV6	

continues

Table 7-1 *Specifications and Standards in the Authentication Framework Domain (Continued)*

Mechanism	Specification	Description
	RFC 2881: Network Access Server Requirements Next Generation (NASREQNG) NAS Model	Proposes a model for NAS—the authenticator
	RFC 2882: Extended RADIUS Practices	
	RFC 2618, 2619, 2620, and 2621	Various RADIUS MIBs
	RFC 2607: Proxy Chaining and Policy Implementation in Roaming	
One-Time Password (OTP)	RFC 2289: A One-Time Password System	
	RFC 2243: OTP Extended Responses	
EAP TLS (EAP Transport Layer Security)	RFC 2716: PPP EAP TLS Authentication Protocol	
EAP TTLS (EAP Tunneled TLS)	draft-ietf-pppext-eap-ttls-03.txt	EAP tunneled TLS authentication protocol
Kerberos	RFC 1510: Kerberos V5	
	RFC 2712: Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)	
	RFC 3244: Microsoft Windows 2000 Kerberos Change Password and Set Password Protocols	
	RFC 3546: TLS Extensions	Updates RFC 2246
	RFC 3268: AES for TLS	
CHAP	RFC 1994: PPP Challenge Handshake Authentication Protocol (CHAP)	
	RFC 2433: Microsoft PPP CHAP Extensions	
	RFC 2759: Microsoft PPP CHAP Extensions, Version 2	
Protected EAP (PEAP)	draft-josefsson-pppext-eap-tls-eap-07.txt	PEAP V2

Table 7-1 *Specifications and Standards in the Authentication Framework Domain (Continued)*

Mechanism	Specification	Description
	draft-kamath-pppext-peapv0-00.txt	Microsoft PEAP version 0 (implementation in Windows XP SP1)
	draft-puthenkulam-eap-binding-04.txt	The compound authentication binding problem
Diameter	RFC 3588: Diameter Base Protocol	
	draft-ietf-aaa-diameter-nasreq-13.txt; Diameter Network Access Server Application	Diameter application in the AAA domain
	draft-ietf-aaa-diameter-cms-sec-04.txt	Diameter CMS security application

EAP

The EAP, a flexible protocol used to carry arbitrary authentication information, is defined in RFC 2284. (Incidentally, RFC 2284 is only 16 pages long!) A set of RFCs also defines the various authentication processes over EAP, including TLS, TTLS, SmartCard, and SIM. The IETF EAP workgroup is working on a revision of the EAP RFC and has submitted the new document as RFC 3579 (was RFC 2284bis).

EAP has two major features. First, it separates the message exchange from the process of authentication by providing an independent exchange layer. By doing so, it achieves the second characteristic: orthogonal extensibility, meaning that the authentication processes can extend the functionality by adopting a newer mechanism without necessarily effecting a corresponding change in the EAP layer.

EAP Frames, Messages, and Choreography

The basic EAP consists of a set of simple constructs: four message types, two message frames, and an extensible choreography.

The four message types are request, response, success, and failure. Figure 7-2 shows the EAP frame format.

As shown in Figure 7-3, EAP also defines a packet to negotiate the EAP protocol configuration. The EAP protocol is identified by C227 (Hex). This packet will be included in the data field of the EAP frame in Figure 7-2.

Figure 7-2 *EAP Frame Format*

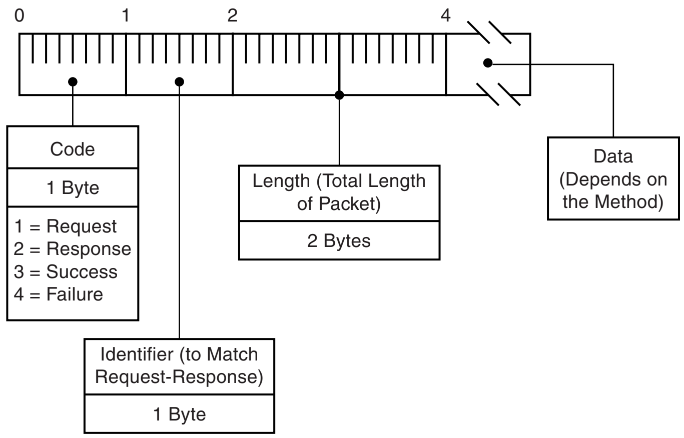
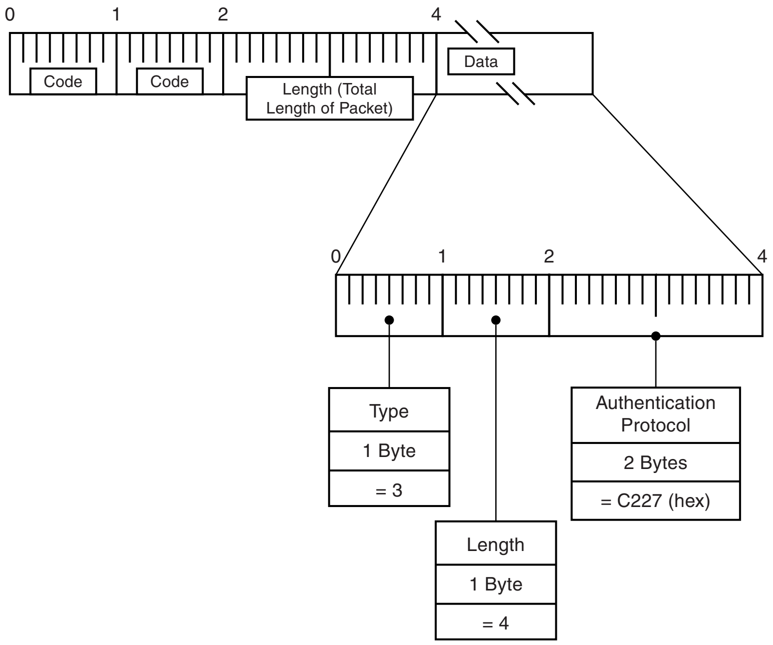
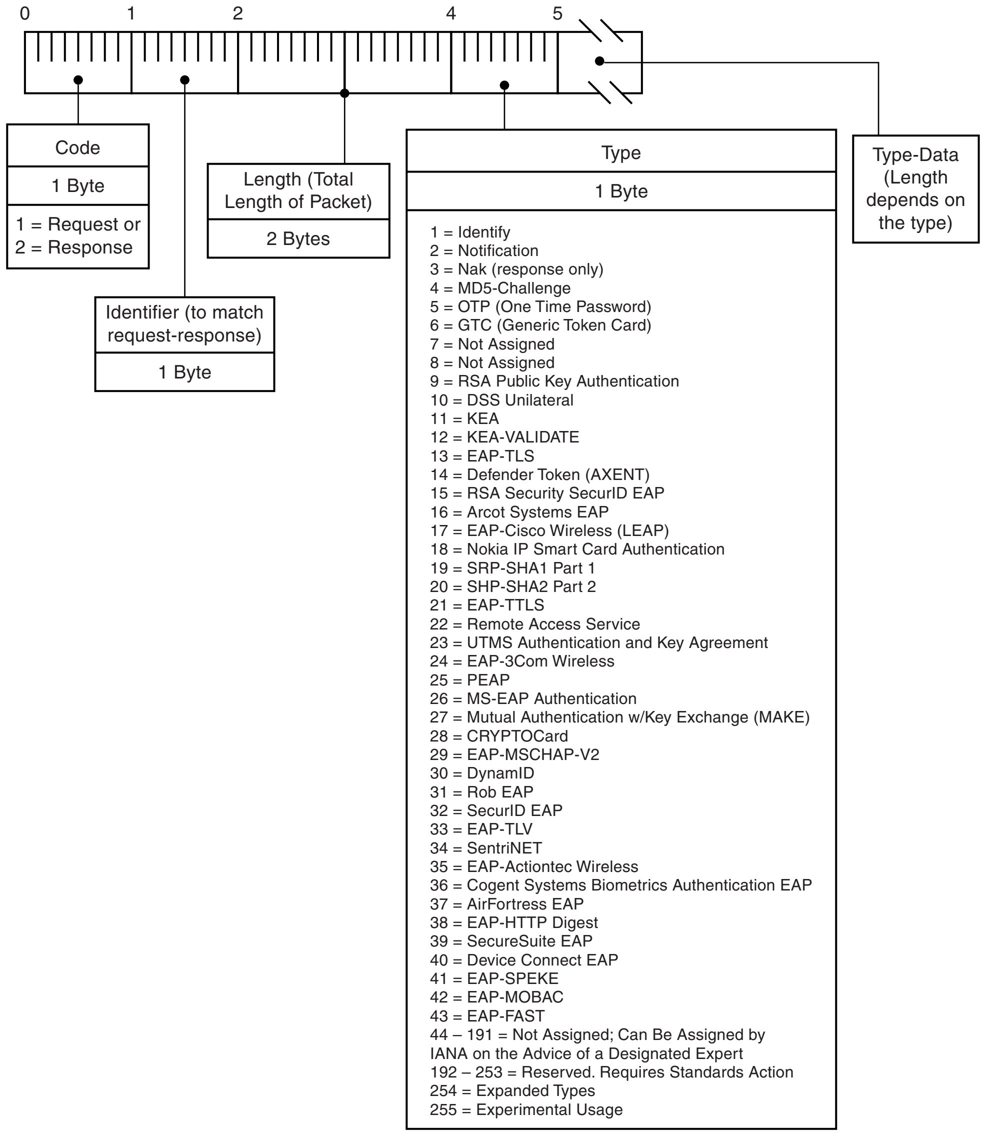

0
1
2
4
Code
1 Byte
1 = Request
2 = Response
3 = Success
4 = Failure
Identifier (to Match Request-Response)
1 Byte
Length (Total Length of Packet)
2 Bytes
Data (Depends on the Method)

Figure 7-3 *EAP Configuration Negotiation Packet*



Depending on the type, the request and response packets include the type field and data, as shown in Figure 7-4.

Figure 7-4 *EAP Request/Response Frame*


0
1
2
4
5
Code
1 Byte
1 = Request or
2 = Response
Identifier (to match request-response)
1 Byte
Length (Total Length of Packet)
2 Bytes
Type
1 Byte
1 = Identify
2 = Notification
3 = Nak (response only)
4 = MD5-Challenge
5 = OTP (One Time Password)
6 = GTC (Generic Token Card)
7 = Not Assigned
8 = Not Assigned
9 = RSA Public Key Authentication
10 = DSS Unilateral
11 = KEA
12 = KEA-VALIDATE
13 = EAP-TLS
14 = Defender Token (AXENT)
15 = RSA Security SecurID EAP
16 = Arcot Systems EAP
17 = EAP-Cisco Wireless (LEAP)
18 = Nokia IP Smart Card Authentication
19 = SRP-SHA1 Part 1
20 = SHP-SHA2 Part 2
21 = EAP-TTLS
22 = Remote Access Service
23 = UTMS Authentication and Key Agreement
24 = EAP-3Com Wireless
25 = PEAP
26 = MS-EAP Authentication
27 = Mutual Authentication w/Key Exchange (MAKE)
28 = CRYPTOCard
29 = EAP-MSCHAP-V2
30 = DynamID
31 = Rob EAP
32 = SecurID EAP
33 = EAP-TLV
34 = SentriNET
35 = EAP-Actiontec Wireless
36 = Cogent Systems Biometrics Authentication EAP
37 = AirFortress EAP
38 = EAP-HTTP Digest
39 = SecureSuite EAP
40 = Device Connect EAP
41 = EAP-SPEKE
42 = EAP-MOBAC
43 = EAP-FAST
44 – 191 = Not Assigned; Can Be Assigned by IANA on the Advice of a Designated Expert
192 – 253 = Reserved. Requires Standards Action
254 = Expanded Types
255 = Experimental Usage
Type-Data (Length depends on the type)

NOTE The RFC assigns eight request/response types. The rest are assigned by the Internet Assigned Numbers Authority (IANA). The current assignments are shown in Table 7-2.

Table 7-2 *EAP Packet Types Assigned by IANA*

Type	Description
1–6	Assigned by RFC
1	Identity
2	Notification
3	Nak (response only)
4	MD5-Challenge
5	One-Time Password (OTP)
6	Generic Token Card (GTC)
7	Not assigned
8	Not assigned
9	RSA Public Key Authentication
10	DSS Unilateral
11	KEA
12	KEA-VALIDATE
13	EAP-TLS
14	Defender Token (AXENT)
15	RSA Security SecurID EAP
16	Arcot Systems EAP
17	EAP-Cisco Wireless (LEAP)
18	Nokia IP SmartCard authentication
19	SRP-SHA1 Part 1
20	SRP-SHA1 Part 2
21	EAP-TTLS
22	Remote Access Service
23	UMTS Authentication and Key Agreement
24	EAP-3Com Wireless

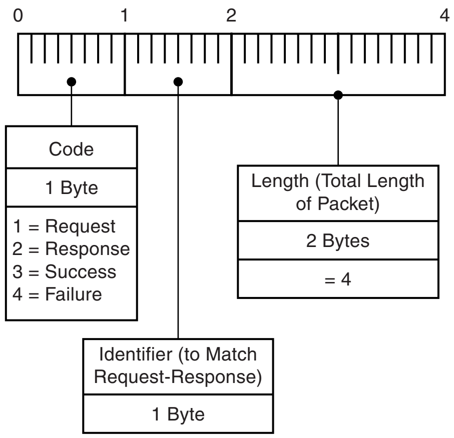
Table 7-2 *EAP Packet Types Assigned by IANA (Continued)*

Type	Description
25	PEAP
26	MS-EAP-Authentication
27	Mutual Authentication w/Key Exchange (MAKE)
28	CRYPTOCard
29	EAP-MSCHAP-V2
30	DynamID
31	Rob EAP
32	SecurID EAP
33	EAP-TLV
34	SentriNET
35	EAP-Actiontec Wireless
36	Cogent Systems Biometrics Authentication EAP
37	AirFortress EAP
38	EAP-HTTP Digest
39	SecureSuite EAP
40	DeviceConnect EAP
41	EAP-SPEKE
42	EAP-MOBAC
43	EAP-FAST
44–191	Not assigned; can be assigned by IANA on the advice of a designated expert
192–253	Reserved; requires standards action
254	Expanded types
255	Experimental usage

NOTE The expanded type (254) frame includes a vendor ID; therefore, it is not deemed interoperable.

Figure 7-5 shows the success/failure frame.

Figure 7-5 *EAP Success/Failure Frame*

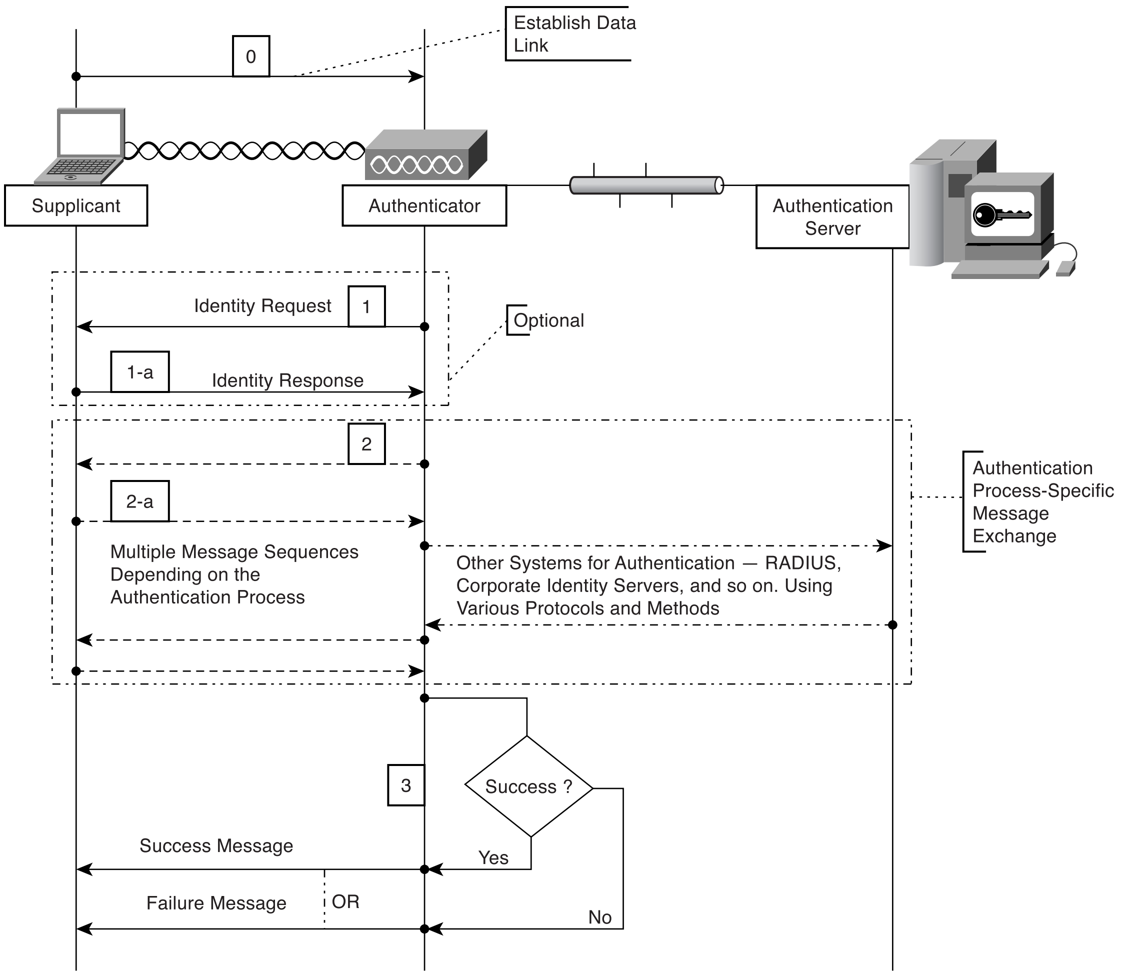


The EAP message exchange is basic, as shown in Figure 7-6. EAP starts after the supplicant has data and link layer connectivity (Step 0 in Figure 7-6). The communication between the authenticator and the supplicant is done as a request-response paradigm, meaning a message is sent and the sender waits for a response before sending another message.

NOTE Generally, either side should be able to start EAP, not just the authenticator. But in this case, notice that the authenticator starts the EAP message, not the supplicant/client. EAP does not assume a specific protocol such as IP, so the messages are "lock-step"—an ordered exchange of messages in which a reply is sent only after receiving the earlier message. Another important observation is that EAP is a point-to-point (peer-to-peer) exchange at the transport layer, not multicast or any other many-to-many mechanism. The choreography is just a minimal framework facilitating further RFCs to define the exact processes. That is what many of the RFCs do: define EAP over various authentication processes such as EAP-SIM, EAP-over-LDAP, EAP-over-GPRS, and of course, EAP-over-802, which is the 802.1x specification.

The first exchange (Step 1 in Figure 7-6) could be an identity exchange. Even though there is an identity message type, the RFC does not guarantee identity semantics and encourages that the authentication mechanisms not depend on this exchange for identity and have their own identity-recognition mechanisms. Moreover, the initial exchange would most likely be in cleartext; therefore, it is a security vulnerability.

Figure 7-6 *EAP Message Exchange Framework*


Establish Data Link
0
Supplicant
Authenticator
Authentication Server
Identity Request
1
Optional
1-a
Identity Response
2
2-a
Multiple Message Sequences Depending on the Authentication Process
Other Systems for Authentication — RADIUS, Corporate Identity Servers, and so on. Using Various Protocols and Methods
Authentication Process-Specific Message Exchange
3
Success ?
Yes
No
Success Message
Failure Message
OR

In Step 2, all the exchanges between the supplicant, authenticator, and back-end authentication systems are defined by a wide variety of specific RFCs or drafts and authentication mechanisms.

Finally, at some point, the authenticator determines whether the authentication is a success or failure and sends an appropriate message to the supplicant (Step 3 in Figure 7-6).

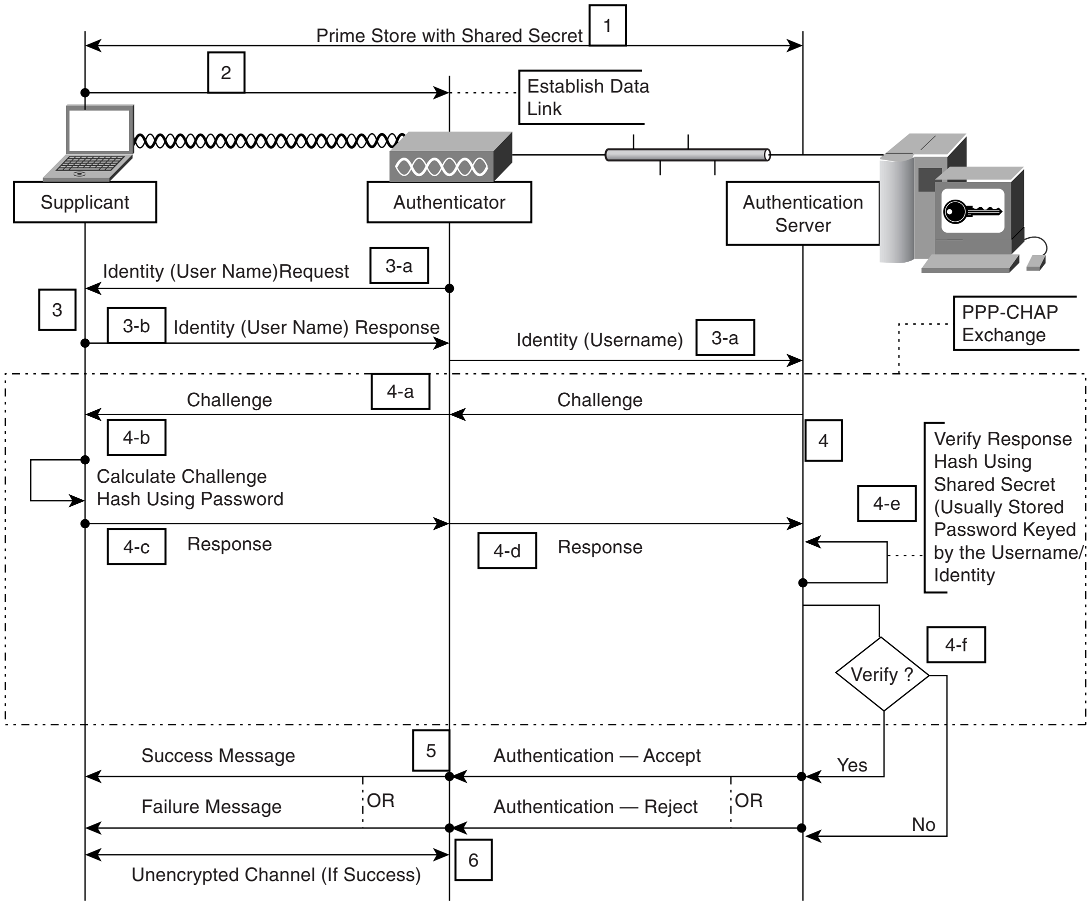
EAP Authentication Mechanisms

This section examines in detail some of the most relevant EAP authentication frameworks. The typical mechanisms using EAP over LANS are EAP-MD5, EAP-One-Time Password (EAP-OTP), EAP-TLS, EAP-TTLS, EAP-Generic Token Card (EAP-GTC), Microsoft CHAP (EAP-MSCHAPv2), and EAP-FAST.

EAP-MD5

The EAP-MD5 is a Challenge Handshake Authentication Protocol (CHAP), as defined in RFC 1994. Figure 7-7 shows the choreography of the EAP-MD5 mechanism.

Figure 7-7 *EAP-MD5 Choreography*



For EAP-MD5 to work, the client and the authentication server must have a shared secret, usually a password associated with an identity/username. This needs to be established out of band (Step 1 in Figure 7-7). The connectivity (Step 2 in Figure 7-7) and identity exchange (Step 3 in Figure 7-7) are required before the EAP-MD5 process. The EAP-MD5 method consists of a random challenge to the supplicant (Step 4-a in Figure 7-7) and a response from the supplicant (Step 4-c, Step 4-d in Figure 7-7), which contains the hash of the challenge created using the shared secret (Step 4-b in Figure 7-7). The authentication server verifies the hash (Step 4-e in Figure 7-7) and accepts or rejects the authentication. The authenticator allows or disallows access (Step 5 in Figure 7-7) based on this decision. If successful, the supplicant gains access (Step 6 in Figure 7-7).

EAP-MD5 is a pure authentication protocol; after the authentication, the messages are transmitted in cleartext. It is also a client authentication protocol—the server side (authenticator) is not authenticated; therefore, it cannot detect a rogue AP.

EAP-MD5 also contains a set of good features: It requires only lightweight processing (which translates to less hardware) and does not require a key/certificate infrastructure. Although pure EAP-MD5 has some value in the PPP world, it is of limited use in the wireless world. For example, Microsoft has dropped the support for EAP-MD5 for the wireless interface in Windows XP. Support was dropped because of security problems; EAP-MD5 is vulnerable to dictionary and brute-force attacks when used with Ethernet and wireless.

EAP-OTP

EAP-OTP is similar to MD5, except it uses the OTP as the response. The request contains a displayable message. The OTP method is defined in RFC 2289. The OTP mechanism is employed extensively in VPN and PPP scenarios but not in the wireless world.

EAP-GTC

The EAP-GTC (Generic Token Card) is similar to the EAP-OTP except with hardware token cards. The request contains a displayable message, and the response contains the string read from the hardware token card.

EAP-TLS

As you have seen, methods such as EAP-MD5 and EAP-GTC are specific to authentication and are confined to authenticating only the client. EAP-TLS adds more capabilities such as mutual authentication, which provides an encrypted transport layer and the capability to dynamically change the keys. On the other hand, EAP-TLS is based on digital certificates and thus requires an infrastructure to manage—issue, revoke, and verify—certificates and keys.

EAP-TLS is based on the TLS protocol that is defined in RFC 2246. The following section talks a little bit about TLS, and then you will look at which of its features carry over into EAP-TLS.

NOTE The origin of the transport level protocol was SSLv1, proposed and implemented by Netscape for securing browser traffic. SSL 1.0 was superseded by SSL 2.0, which was the original SSL. SSL 3.0, which, of course, superseded SSL 2.0, is the most common security protocol used today. IETF chartered a working group in 1996, accepted submissions from Netscape (SSL 3.0) and Microsoft (PCT), and delivered RFC 2246—TLS 1.0.

A Brief Introduction to TLS

TLS has the concept of sessions and connection. A connection is a channel, whereas a session is governed by security context—session identifier, peer certificate, compression method, cipher spec for the session key, and MAC algorithm parameters and the shared master secret. TLS can and will securely negotiate different session parameters while maintaining the same connection—usually a TCP connection. The handshake phase establishes a session, and the session keys (symmetric) encrypt the transport during the data transfer phase. In addition to providing confidentiality, TLS provides integrity check. TLS, of course, is a point-to-point method.

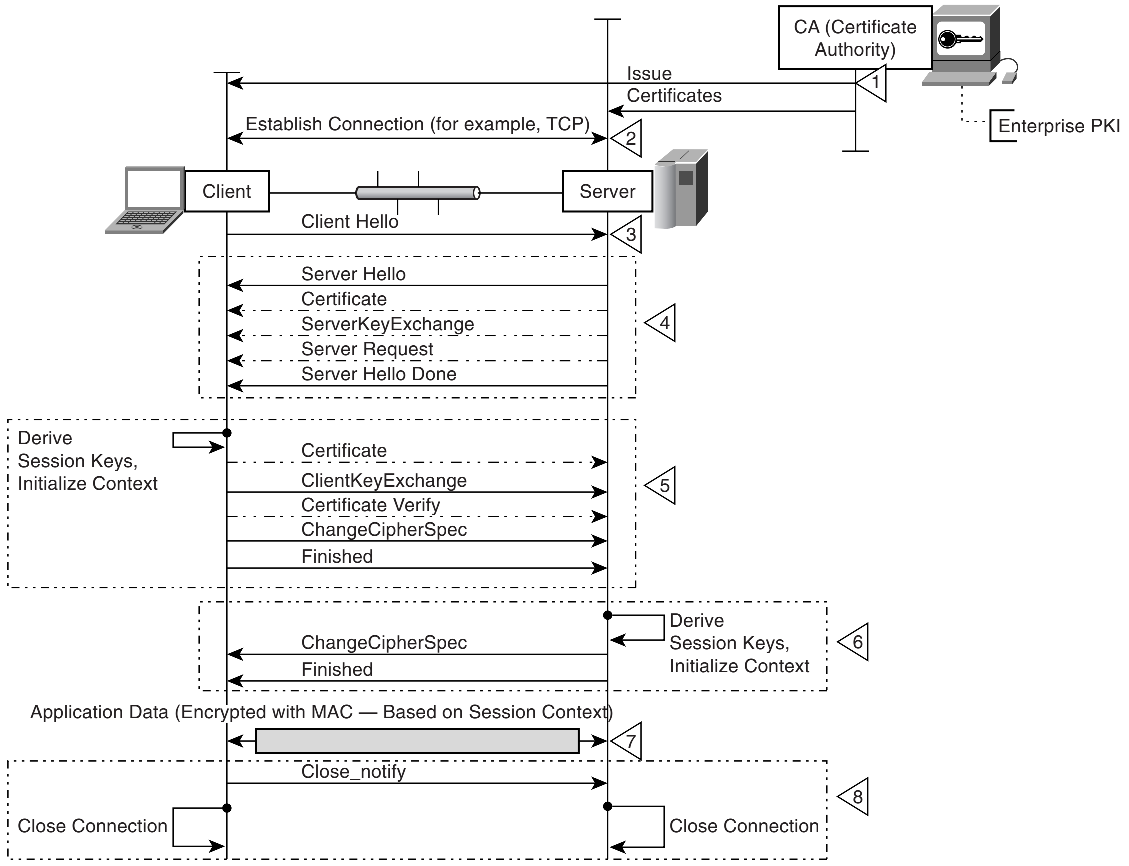
TLS defines two layers: a record layer (which exchanges messages dealing with things such as fragmentation, MAC, and encryption) and a message layer (which defines different types of messages). The four message types are as follows:

- **Change cipher spec**—Used to signify change in the session context to be used by the record layer. This is an independent content type that is used to avoid getting trapped in specific protocol messages, at which point the pipe could stall.
- **Alert**—Could be warning or fatal. The alert message subtypes (approximately 26 sub-types) include close notify, decryption failed, certificate revoked, access denied, and so on.
- **Handshake protocol**—You will see these messages in Figure 7-8. The subtypes include the following:
  - Hello messages (hello_request, client_hello, and server_hello)
  - Server authentication and key exchange messages (certificate, server_key_exchange, certificate_request, and server_hello_done)
  - Client authentication and key exchange messages (certificate_verify and client_key_exchange)
  - Handshake finalization message (finished)

- **Application data**—The records themselves are transmitted over a reliable protocol such as TCP. TLS also defines a handshake protocol for authentication, exchanging cryptographic parameters and establishing session context.

Figure 7-8 shows the TLS choreography, through the lifetime of a connection, in some detail.

Figure 7-8 *TLS Choreography*



The handshake protocol (Steps 3, 4, 5, and 6 in Figure 7-8) accomplishes server authentication, algorithm negotiation, establishing session context, and (optional) client authentication. Of course, to successfully complete the handshake and arrive at the keys and secrets, the client and server should have digital certificates (Step 1 in Figure 7-8) and connectivity (Step 2 in Figure 7-8).

After the handshake is successfully completed, the client and server can exchange application data (Step 7 in Figure 7-8) using the established secure transport. Occasionally, renegotiation of session context might happen, usually for new session keys. Finally, the client or server with the close message closes the connection (Step 8 in Figure 7-8).

EAP-TLS Choreography

EAP-TLS employs selected parts of the TLS. For example, it uses the TLS handshake for mutual authentication, cipher suit negotiation, and to derive session keys; however, it does not use all parts of the TLS record protocol.

Figure 7-9 *EAP-TLS Frame Format*

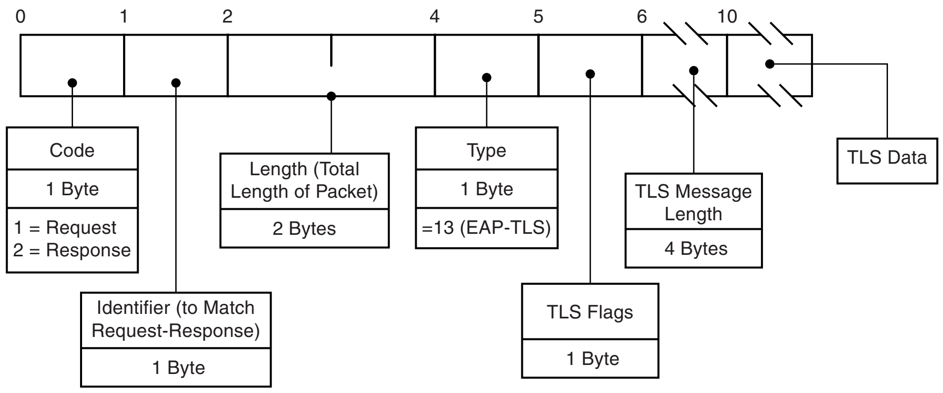
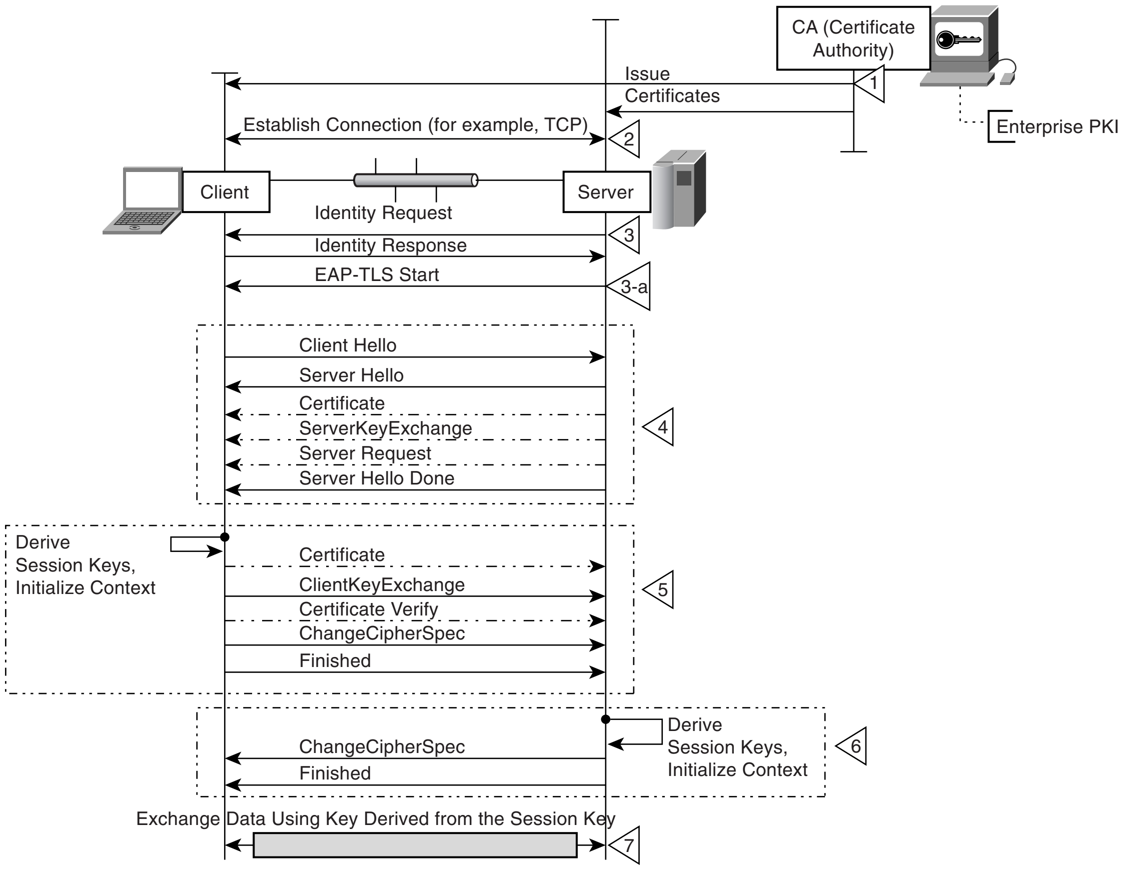


Figure 7-9 shows the frame format for EAP-TLS. The EAP type is 13 (see Table 7-2). The EAP data frame consists of TLS-specific fields. A similar approach is taken for the choreography, as shown in Figure 7-10. As expected, Figure 7-10 is a combination of Figures 7-6 and 7-8.

After the EAP identity request and response, a TLS-START request is sent (this is where Bit 2 of the TLS flag is used) to the supplicant (Step 3-a in Figure 7-10). This initiates the TLS handshake protocol (remember, TLS starts with a client-hello), which, in the end, results in authentication and establishing session keys for securing (confidentiality and integrity) the transport layer. As you saw in the TLS section, the session context contains all the relevant information. After the handshake is done, EAP-TLS does not use any of the TLS record protocols; that is, the application data is not exchanged using the TLS record protocol.

Figure 7-10 *EAP-TLS Choreography*



NOTE As you can see, this still does not satisfy all the requirements of the wireless world because EAP-TLS is written for PPP, where a key and authentication are sufficient for communication and the client authenticates the server. The wireless world also has the authenticator/AP, which has to be authenticated, and the server is a RADIUS server.

In the wireless world, EAP-TLS is used as a strong and secure means for authentication and key establishment. After that, the native WEP mechanisms are used to encrypt the data. In fact, the WEP encryption key is derived from the TLS session key.

EAP-TTLS

EAP-TTLS is similar to EAP-TLS, but the client authentication is extended after the secure transport has been established. Then the client can be authenticated using any of the methods like username/PW, CHAP, and MSCHAPv2. This is called *tunneled authentication*. What this achieves is that the client does not require a digital certificate; only the authentication server needs one. This capability simplifies the client credential management. Organizations can also use currently available/legacy authentication methods (usually password-based schemes).

PEAP

In many ways, PEAP is actually EAP over TLS for the wireless domain. In this section, you will see how PEAP adds capabilities needed in the wireless domain, such as chaining EAP mechanisms and exchange of arbitrary parameters, cryptographic binding between EAP mechanism and the tunnel, session optimization, and generic reauthentication.

From a draft perspective, all the EAP drafts are generic and do not fully address the wireless domain. In addition, RFC 3579 is superseding RFC 2284. The PEAP draft aims at providing secure EAP authentication for 802.11 based on the new EAP drafts.

NOTE One of the major security vulnerabilities from the EAP perspective is that some of the outer/initial exchanges, such as identity and results, are sent in the clear. This can result in denial-of-service (DoS) vulnerability; for example, an intruder can flood EAP failure messages. Inner exchanges such as EAP-MD5, EAP-SIM, and EAP-MSCHAPV2 also are not fully and uniformly protected. In many cases, the credential exchanges are open to attacks, such as dictionary attacks on a password.

The opportunity for vulnerability is complicated by the "compound binding problem" with PEAP and like protocols, in which two otherwise-secure protocols are combined without cryptographic handoff and might become less secure in combination than separate. On the other hand, password-based EAP protocols are simpler to manage.

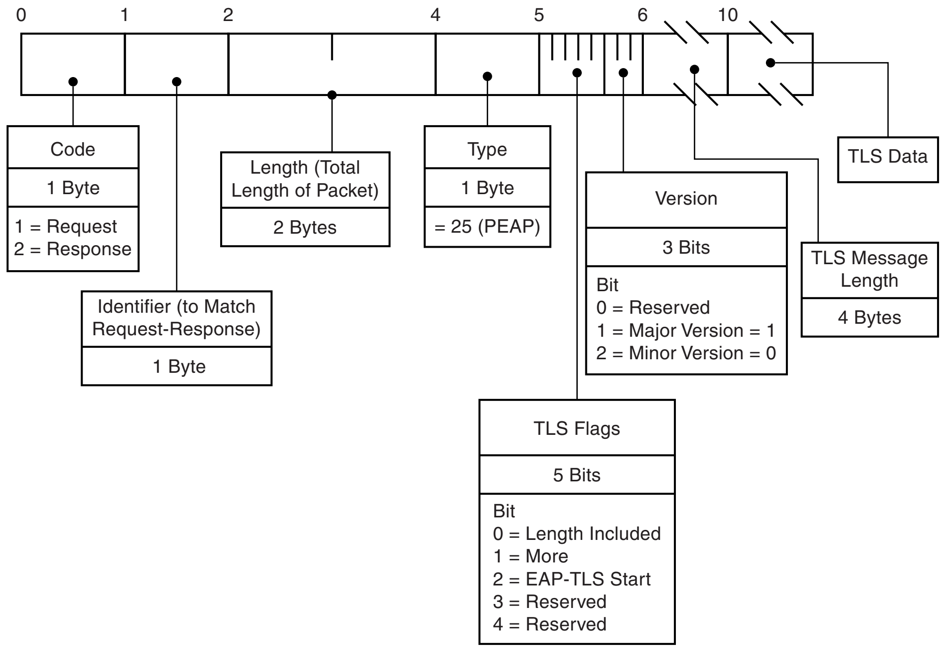
PEAP aims at leveraging EAL-TLS, securing the open exchanges, and facilitating any of the EAP mechanisms over the secure channel, thus maintaining the simplicity (as far as possible) with the required level of security. For example, PEAP requires only server-side certificates, uses TLS for the secure tunnel, and extends the EAP-TLS beyond the finished message exchange to add client authentication and key exchange. The client authentication can be any of the EAP methods and thus can achieve security and the use of existing authentication paradigms. Of course, PEAP has some drawbacks. It is a little chatty (because of more message exchanges) and does require a certificate infrastructure for the servers. Also, TLS is normally implemented over a reliable transport-TCP, so implementing TLS over EAP requires small reliability and retransmit mechanisms.

The PEAP protocol has two phases. The first phase is to establish a secure tunnel using the EAP-TLS with server authentication. The second phase implements the client authentication based on EAP methods, exchange of arbitrary information, and other PEAP-specific capabilities through the secure transport established during phase 1. It will be instructive to see how PEAP manages to stay within EAP-TLS (for the most part), still adding capabilities. This is important to achieve a simpler supporting infrastructure.

PEAP Frame Format

Figure 7-11 shows the PEAP request and response format.

Figure 7-11 *PEAP Frame Format*



The PEAP frame format is almost the same as the EAP-TLS format, the difference being the version bits in the flags field and the type (25 for PEAP versus 13 for EAP-TLS; see Table 7-2).

PEAP Arbitrary Parameter Exchange

The type-length-value (TLV) mechanism is used to exchange arbitrary name-value pairs. Because this exchange happens in the second phase of the PEAP exchange, the frame formats are EAP formats with type 33 (see Table 7-2) but different from the TLS domain.

Figure 7-12 shows the frame format for the TLV mechanism. The RFC has defined approximately eight TLV types. Figure 7-13 shows the vendor-specific TLV. As you can see, this makes PEAP totally extensible but specific to a vendor.

Figure 7-12 *PEAP TLV Frame Format*

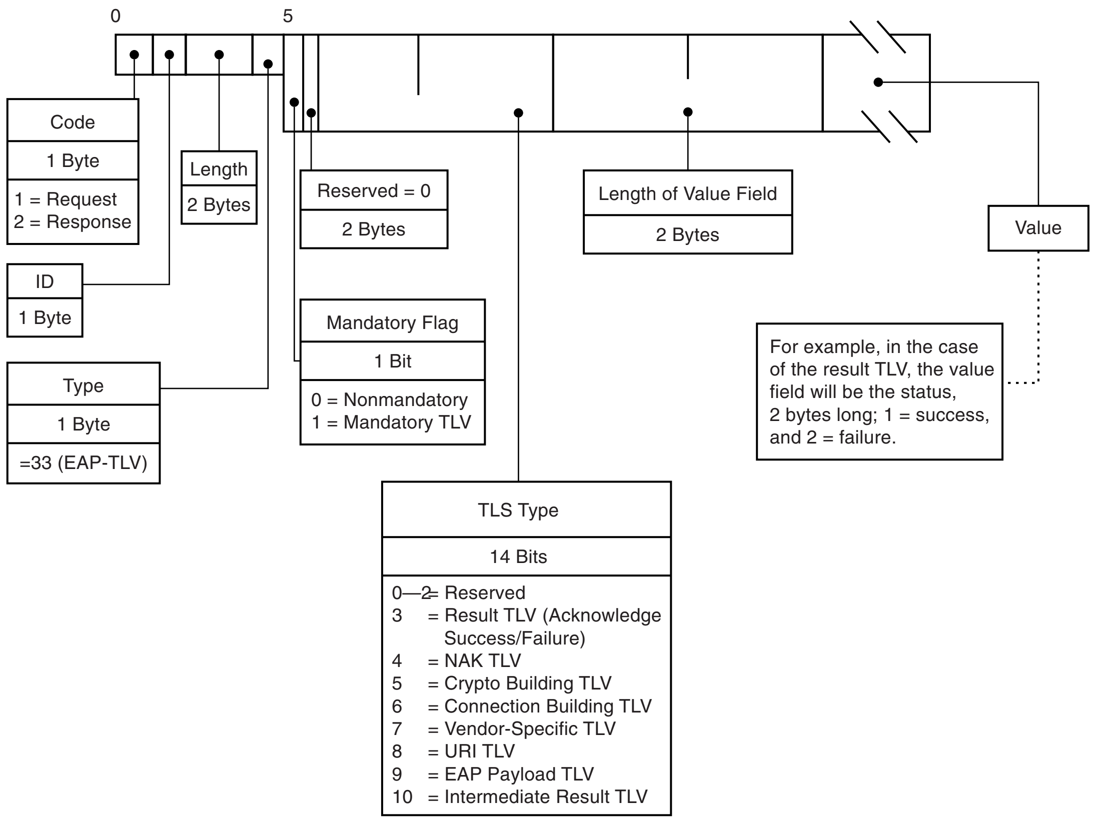
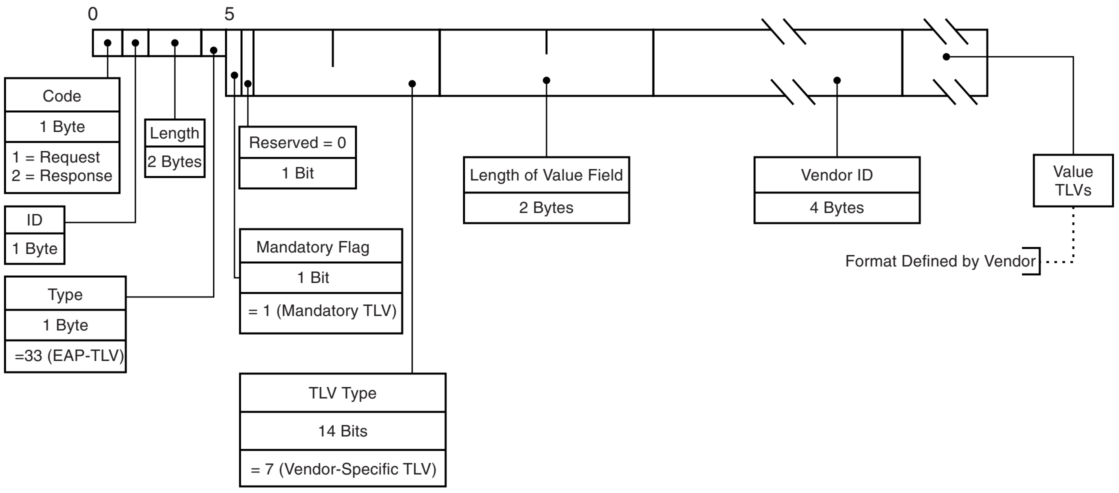
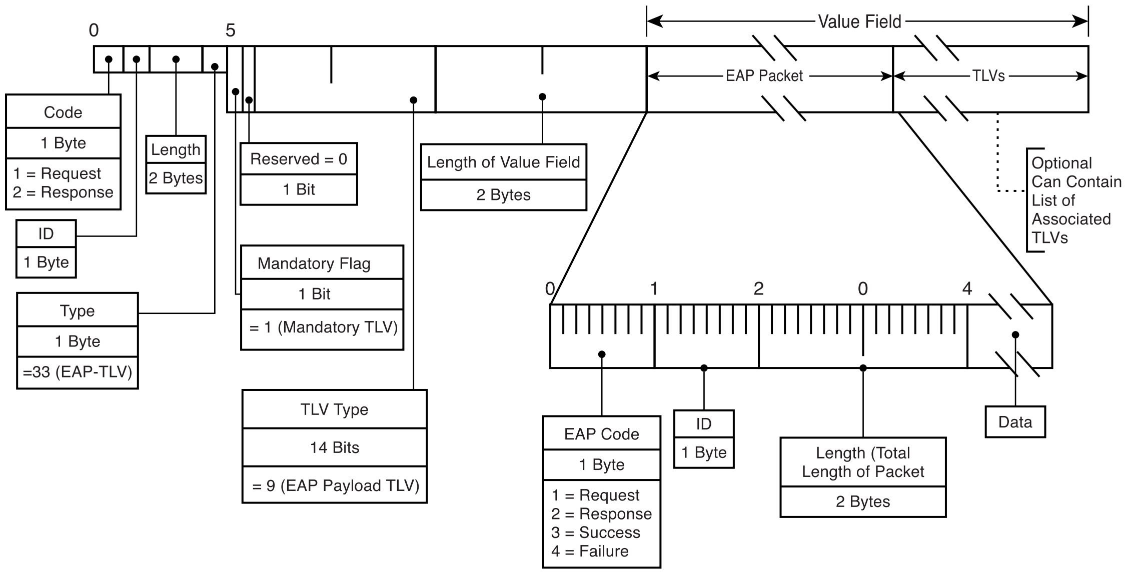


Figure 7-13 *PEAP TLV Frame Format—Vendor-Specific TLV*



Another interesting mechanism is the EAP Payload TLV shown in Figure 7-14, which encapsulates the EAP frame in a PEAP-TLV frame. This is powerful because it can tunnel EAP methods over the secure transport. The following subsection shows how this is being used in the PEAP phase 2 choreography.

Figure 7-14 *PEAP TLV Frame Format—EAP Payload TLV*



NOTE Not all PEAP implementations are required to understand all the TLV types. The mandatory flag indicates this disposition. The mandatory TLV types are the EAP Payload TLV, Intermediate Result TLV, vendor-specific TLV (syntactical—that is, it should understand that it is a vendor-specific TLV; semantic understanding depends on the vendor implementation), Result TLV, and NAK TLV. The NAK TLV is used to indicate if an entity cannot understand the syntax of a TLV.

Another feature in the specs is the optimization of TLV message exchange; the spec allows multiple TLVs to be sent in one message—the only caveat being that multiple TLVs in one message are not allowed for the EAP Payload TLV.

PEAP Choreography

The PEAP choreography is similar (in fact, the same in most of the cases) to EAP-TLS. The main difference is that PEAP does not require client authentication, and the message exchange extends beyond where EAP-TLS stops.

Figure 7-15 shows the PEAP exchange.

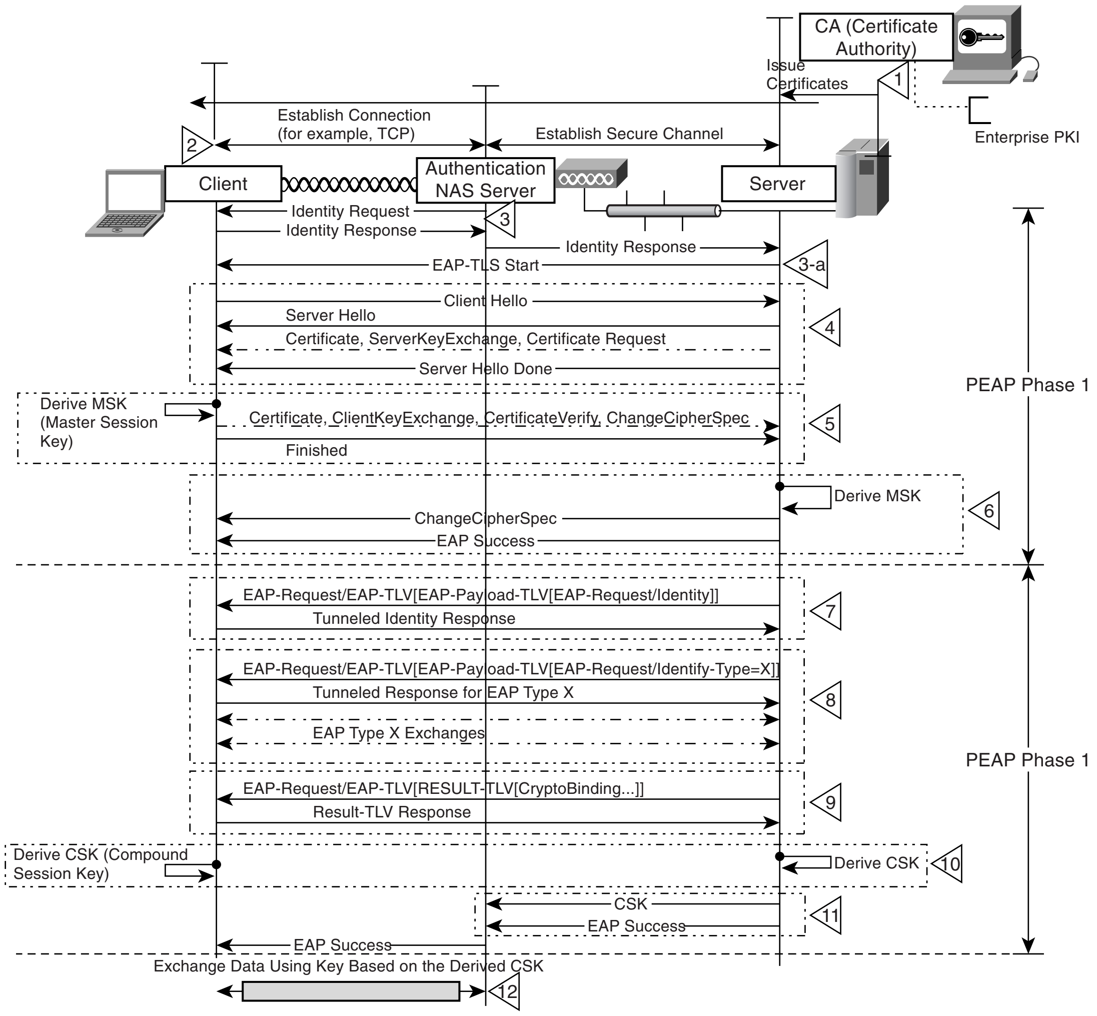
As you can see, the PEAP conversation is between the EAP server and the EAP peer, and the authenticator acts as a pass-through for most of the conversation. The advantage of this scheme is that newer EAP schemes can be developed and implemented without changing the authenticator and NAS—only the peer(supplicant/client) and the EAP server need to be updated. This results in easier and simpler upgrade to the supporting infrastructure.

Step 1 Similar to EAP-TLS, the EAP server requires a certificate; the client/peer certificate is optional.

Step 2 The client/peer must establish a connection to the authenticator—in this case, a wireless connection. An important requirement is the secure channel between the authenticator and the EAP server. This is vital because the specification does not indicate how this is established, but it requires one.

Step 3 The identity request-response is the basic EAP sequence, which is sent in the clear. In PEAP, this is used for administrative purposes, such as which server to select, and possibly for other initial context setup. The identity, which is sent in the clear, should not be used for any other purposes. Any identity exchange should happen in phase 2 after the secure tunnel is established—for example, tunneling the identity request-response using the EAP-TLV mechanism (Step 7). The identity response is sent to the EAP server, which in turn starts the process with the EAP-TLS start message.

Figure 7-15 *PEAP Choreography*


CA (Certificate Authority)
Issue Certificates
1
Enterprise PKI
Establish Connection (for example, TCP)
Establish Secure Channel
2
Client
Authentication NAS Server
Server
Identity Request
Identity Response
3
Identity Response
EAP-TLS Start
3-a
Client Hello
Server Hello
Certificate, ServerKeyExchange, Certificate Request
Server Hello Done
4
Derive MSK (Master Session Key)
Certificate, ClientKeyExchange, CertificateVerify, ChangeCipherSpec
Finished
5
Derive MSK
ChangeCipherSpec
EAP Success
6
PEAP Phase 1
EAP-Request/EAP-TLV[EAP-Payload-TLV[EAP-Request/Identity]]
Tunneled Identity Response
7
EAP-Request/EAP-TLV[EAP-Payload-TLV[EAP-Request/Identify-Type=X]]
Tunneled Response for EAP Type X
EAP Type X Exchanges
8
PEAP Phase 1
EAP-Request/EAP-TLV[RESULT-TLV[CryptoBinding...]]
Result-TLV Response
9
Derive CSK (Compound Session Key)
Derive CSK
10
CSK
EAP Success
11
EAP Success
Exchange Data Using Key Based on the Derived CSK
12

Steps 4, 5, and 6 These steps are typical EAP-TLS exchanges. Usually the client certificate is not exchanged. The successful completion of the EAP-TLS ends phase 1, and phase 2 leverages the secure tunnel created by phase 1.

Step 7 This is the beginning of phase 2. The EAP-TLV mechanism can be used to tunnel the normal EAP identity exchange.

Step 8 In this step, the EAP server authenticates the client using any of the EAP mechanisms: EAP-MD5, EAP-CHAP, EAP-SIM, and so on. The exchange is fully protected by the TLS tunnel, and the EAP-TLV choreography allows a graceful mechanism to affect the EAP mechanisms. This is the heart of the PEAP method—the server with a certificate, the establishment of the tunnel by TLS, and the use of the EAP methods available in the organization's infrastructure.

Step 9 This is the final stage of crypto binding and so on between the client and the EAP server.

Step 10 In this step, the client and the server derive the required keys.

Key Derivation, Exchange, and Management

The description in this section really skipped over the more intimate details about key derivation, exchange, and management. You should read the PEAP RFC for the details; there are key derivation algorithms, key management sequences, and theory. The Compound Session Key (CSK) is actually a concatenation of the Master Session Key (MSK), which is 64 bytes, and the Extended Master Session Key (EMSK), which is 64 bytes.

The MSK and EMSK are defined in RFC 3269 (also known as RFC 2284bis) as follows:

- **Master Session Key**—Key derived between the peer and the EAP server and exported to the authenticator.
- **Extended Master Session Key**—Additional keying material derived between the peer and the EAP server and exported to the authenticator. It is reserved for future use and not defined in the current RFC. In addition, the PEAP key mechanisms are designed for future extensibility; the exchange sequences (and choreographies) and formats can be used for handling any key material; binding inner, outer, and other intermediate methods; and verifying the security between the layers that are required for future algorithms.

Step 11 This is where the authenticator receives the keys and the result of the authentication process.

Step 12 Now the client and AP can exchange information using the keys that are derived from the PEAP mechanism.

There are a lot more details and capabilities, such as reauthentication using the session resumption feature of TLS, fast roaming, fragmentation and assembly, key rotation and rekeying, and so on, in PEAP. In short, PEAP is a powerful mechanism that is in its initial stages of implementation.

802.1x: Introduction and General Principles

As you have seen, the EAP and other methods are primarily developed for dial-up connections; therefore, there are no link layer protocols for them in the 802 LAN worlds. You cannot arbitrarily open up a TCP port and start sending EAP data. That is where 802.1x comes in. It provides a set of context (such as port and supplicant), state machines between the various layers, and the EAP over LAN (EAPOL) protocol. Of course, 802.1x is not specific to WLANS; in fact, the standard is being used in wired networks successfully. 802.1x provides the access models, whereas EAP adds the authentication mechanisms.

NOTE The 802.1x specification is clear about what 802.1x does and does not do. It provides a framework but does not specify the information (credentials and other challenge-response artifacts) or the basis of authentication (such as how to authenticate, what information is used to authenticate, how the decisions are made, and what authorizations are allowed as a result of the authentication).

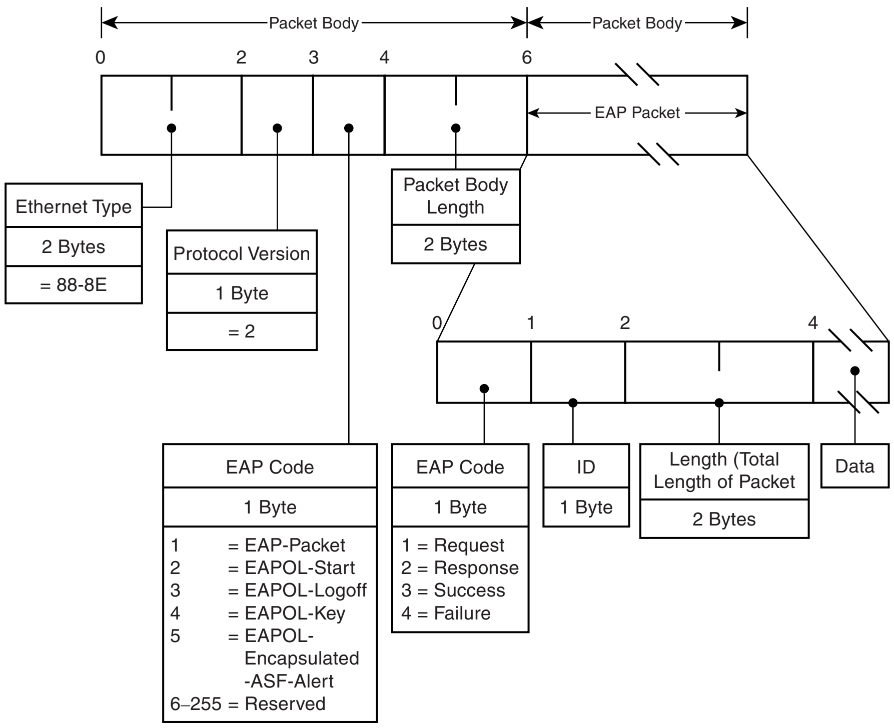
The 802.1x specification starts with the concept of a port as single entry into a network for a supplicant. Hence, it covers 802.3 networks while considering a shared medium like the classical token ring out of scope. In fact, the 802.1x defines EAPOL only for 802.3 Ethernet MACs and Token Ring/FDDI MACs. As previously shown, this plays well with the 802.11 in which each client can be associated with only one AP; hence, the connection to an AP is analogous to the port in the 802.1x realm.

A controlled port is one that allows access after a successful authentication. A controlled port probably offers all the network services. The concept of an uncontrolled port also exists and is important because initial messages and authentication services would be offered through an uncontrolled port. Usually only minimal administrative services are offered by an uncontrolled port.

EAPOL

EAP encapsulation over LAN (EAPOL) is the method to transport EAP packets between a supplicant and an authenticator directly by a LAN MAC service. Figure 7-16 shows the MAC Protocol Data Unit (MPDU) for Ethernet. The header fields include Ethernet type, protocol version, packet type, and body length.

Figure 7-16 *EAPOL MPDU for 802.3/Ethernet*



The body itself is the EAP packet you saw in earlier sections dealing with EAP.

NOTE For the Token Ring/FDDI, the MPDU header is 12 bytes long with the first field SNAP-encoded Ethernet type.

As you might have guessed by now, a supplicant can initiate an authentication by the EAPOL-start frame. But usually a port in an authenticator becomes active (by a connection from a client), and the authenticator starts the EAP process, usually by an EAP-request-identity message encapsulated as EAP type in the EAPOL packet type field. One important packet type is the EAPOL-logoff from a supplicant to the authenticator. In the 802.11 world, this ends an association.

802.1x deals extensively with state machines, timers, handoff between the various layers, and port access control MIBs for SNMP. You can best understand these concepts by reading the standard.

Cisco LEAP (EAP-Cisco Wireless)

Cisco LEAP was developed at a time when WEP showed vulnerabilities and the full wireless security blueprint was not standardized. Moreover, instead of requiring a certificate infrastructure for clients, organizations wanted to leverage authentications that were already available within their infrastructure for secure WLAN. So Cisco developed a lightweight protocol that leveraged many of the existing features and still provided the required security features.

LEAP uses 802.1x EAPOL messages, performs server authentication, achieves username/password (over MS-CHAP) as the user authentication mechanism, uses a RADIUS server as the authentication server, and provides mechanisms for deriving and distributing encryption keys.

NOTE The EAP type is EAP-Cisco Wireless (see Table 7-2).

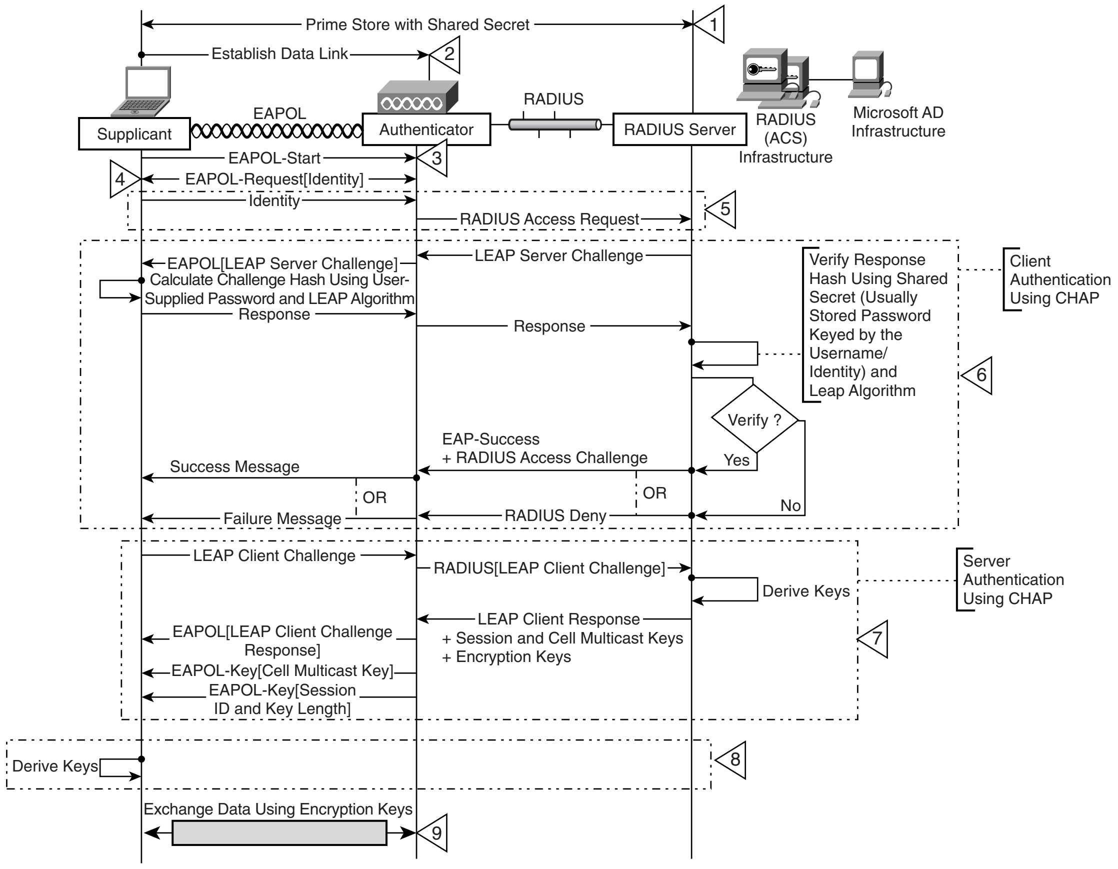
Figure 7-17 details the LEAP choreography.

The entities that participate in a LEAP exchange are the RADIUS server, the AP, and the client.

In Step 1, the client and the RADIUS server should have the shared secret, usually a username-password database of all users in the RADIUS server (or access to a Microsoft Active Directory infrastructure), and each client should have its own username and password.

After a client establishes connectivity (Step 2), it initiates the authentication process by an EAPOL-start (Step 3), to which the AP responds by an EAP-request-identity message over EAPOL (Step 4).

Figure 7-17 *LEAP Choreography*



Prime Store with Shared Secret
1
Establish Data Link
2
EAPOL
RADIUS
Supplicant
Authenticator
RADIUS Server
RADIUS (ACS) Infrastructure
Microsoft AD Infrastructure
EAPOL-Start
3
4
EAPOL-Request[Identity]
Identity
RADIUS Access Request
5
EAPOL[LEAP Server Challenge]
LEAP Server Challenge
Calculate Challenge Hash Using User-Supplied Password and LEAP Algorithm
Response
Response
Verify Response Hash Using Shared Secret (Usually Stored Password Keyed by the Username/ Identity) and Leap Algorithm
Client Authentication Using CHAP
6
Verify ?
EAP-Success + RADIUS Access Challenge
Yes
Success Message
OR
OR
No
Failure Message
RADIUS Deny
LEAP Client Challenge
RADIUS[LEAP Client Challenge]
Derive Keys
Server Authentication Using CHAP
LEAP Client Response + Session and Cell Multicast Keys + Encryption Keys
EAPOL[LEAP Client Challenge Response]
7
EAPOL-Key[Cell Multicast Key]
EAPOL-Key[Session ID and Key Length]
Derive Keys
8
Exchange Data Using Encryption Keys
9

The client response with identity is sent to the RADIUS server in a RADIUS message (Step 5).

From this point on, the AP acts as a relay between the client and the RADIUS server, until after Step 7.

Step 6 is client authentication by challenge-response mechanism. The server sends a challenge, to which the client responds with a hash calculated using the password and the LEAP algorithm. The server also calculates the hash, and if they are equal, the authentication is success. As you can see, the client authentication happens based on existing infrastructure and still not transmitting the credential (here the password).

In Step 7, the server authentication happens through a similar mechanism, and at the end, the server sends the encryption keys to the AP. The AP distributes the required key material by broadcast.

The client derives the encryption key from the key materials (Step 8), and from then on, the AP and the client can use the encryption keys to have a secure conversation (Step 9).

NOTE The LEAP key generation mechanism is proprietary and is generated every (re)authentication, thus achieving key rotation. The session timeout in RADIUS allows for periodic key rotation, thus achieving security against sniffing and hacking the keys. The RADIUS exchanges for LEAP include a couple of Cisco-specific attributes in the RADIUS messages.

EAP-FAST

Comparing the various methods, the EAP-FAST mechanism is the most comprehensive and secure WLAN scheme. LEAP was proven to be susceptible to dictionary attacks, and EAP-FAST is preferable to LEAP. In short, EAP-FAST is hardened LEAP with better crypto protecting the challenge/response mechanism.

EAP-FAST not only mitigates risks from passive dictionary attacks and man-in-the-middle (MitM) attacks, it also enables secure authentication based on currently deployed infrastructure. In addition, EAP-FAST minimizes the hardware requirement; many of the mechanisms require computational burden at the edge devices for asymmetric cryptography and certificate validation. As you have seen from your experience, secure-but-difficult-to-deploy mechanisms would not be popular; hence, EAP-FAST's features (such as flexible deployment model, support for secure provisioning, and efficiency) make it attractive for deployments.

NOTE EAP-FAST started out as Tunneled EAP (TEAP), also known as LEAP V2. But as it evolves, it has become more than a LEAP replacement and is maturing. The final specification might be a little different from what is portrayed here, but the major concepts will not be different.

EAP-FAST is available as an informational Internet draft at http://www.ietf.org/internet-drafts/draft-cam-winget-eap-fast-00.txt.

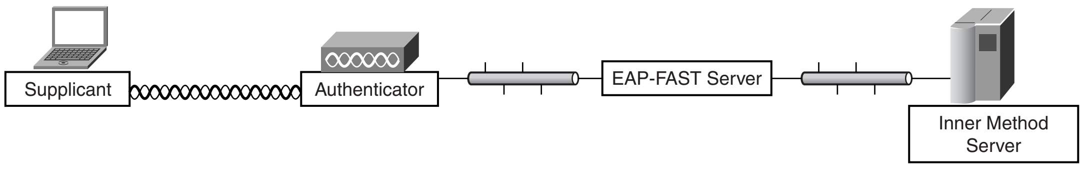
To bootstrap the process securely, EAP-FAST establishes a shared secret (between the client and the authentication server) referred to as the *Protected Access Credential Key (PAC-Key)*. The PAC consists of the PAC-Key (32 bytes), an opaque field cached by the server, and PAC info (metadata about the PAC). The PAC is used to establish a tunnel that is then used to perform authentication. The three-phase EAP-FAST protocol is shown in Table 7-3.

Table 7-3 *EAP-FAST Phases*

Phase	Function	Description	Purpose
Phase 0	In-band provisioning—provide the peer with a shared secret to be used in secure phase 1 conversation	Uses Authenticated Diffie-Hellman Protocol (ADHP) This phase is independent of other phases; hence, any other scheme (in-band or out-of-band) can be used in the future.	Eliminate the requirement in the client to establish a master secret every time a client requires network access
Phase 1	Tunnel establishment	Authenticates using the PAC and establishes a tunnel key	Key establishment to provide confidentiality and integrity during the authentication process in phase 2
Phase 2	Authentication	Authenticates the peer	Multiple tunneled, secure authentication mechanisms

Figure 7-18 shows the functional entities involved in an EAP-FAST exchange. Of course, more than one function can be embedded in one server or software layer.

Figure 7-18 *EAP-FAST Functional Entities*

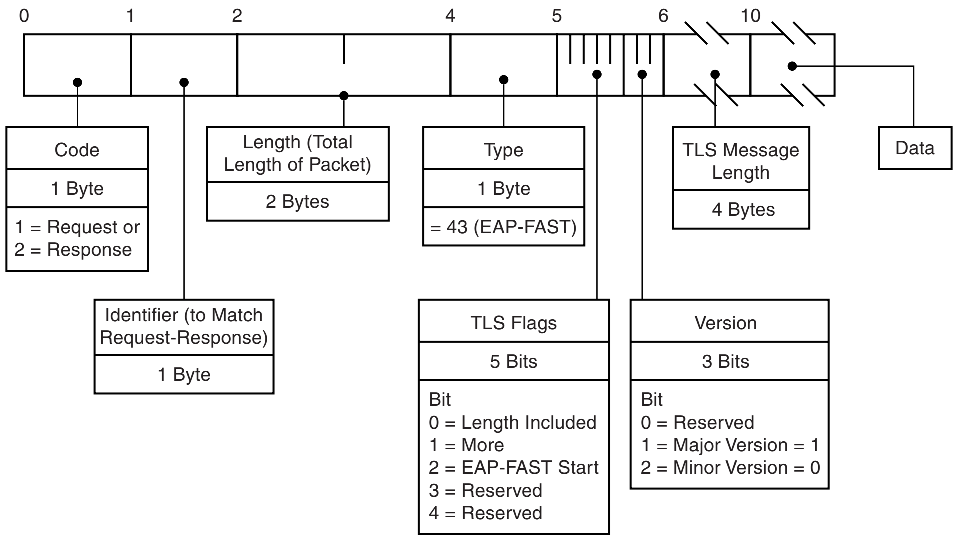


NOTE The separation of duties between an EAP-FAST server and the inner method server adds deployment flexibility and extensibility. An organization can use the current and available authentication infrastructure and then progressively move to any other infrastructure it chooses.

EAP-FAST Frame Format

As shown in Figure 7-19, the EAP-FAST frame format is similar to the TLS format for phase 1.

Figure 7-19 *EAP-FAST Frame Format*



The major contribution by EAP-FAST to the frame format is the PAC fields and associated information in the phase 0 and subsequent conversations. Figure 7-20 shows the PAC-TLV.

Table 7-4 describes some of the salient fields.

EAP-FAST Choreography

The EAP-FAST choreography is a combination of multiple conversations. Figure 7-21 shows an overview of the EAP-FAST choreography.

Figure 7-20 *PAC TLV Frame Format*

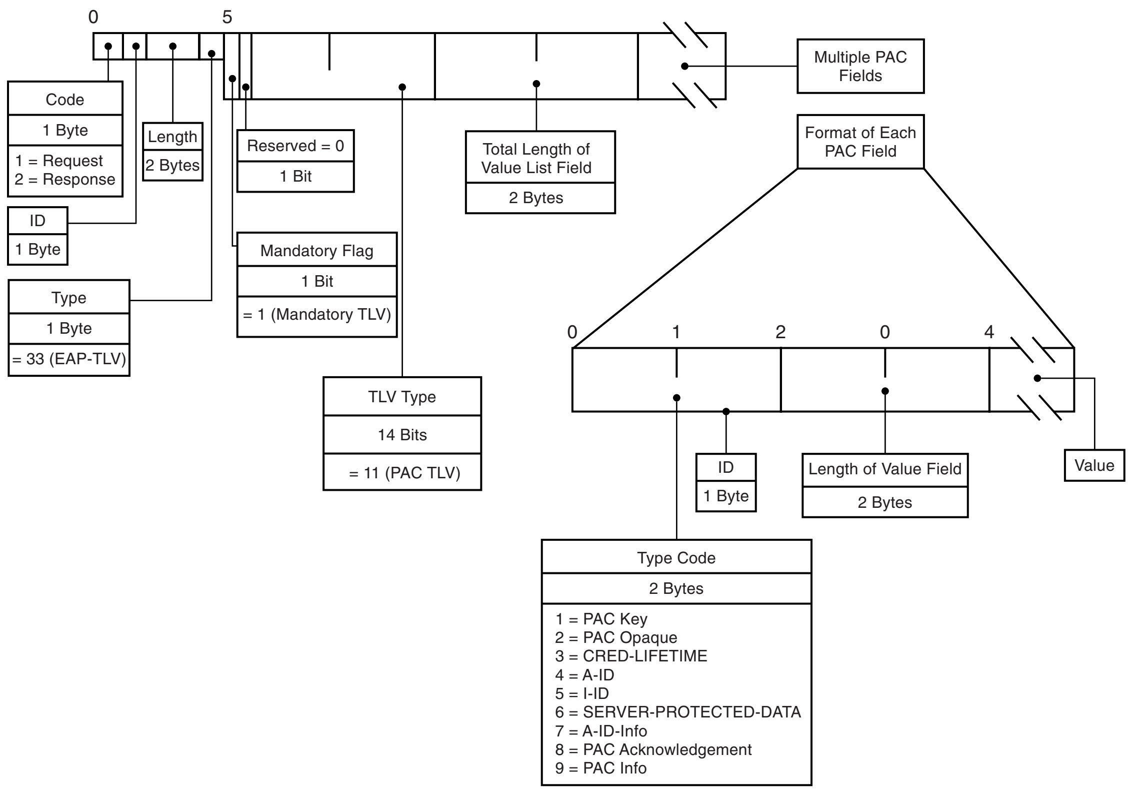
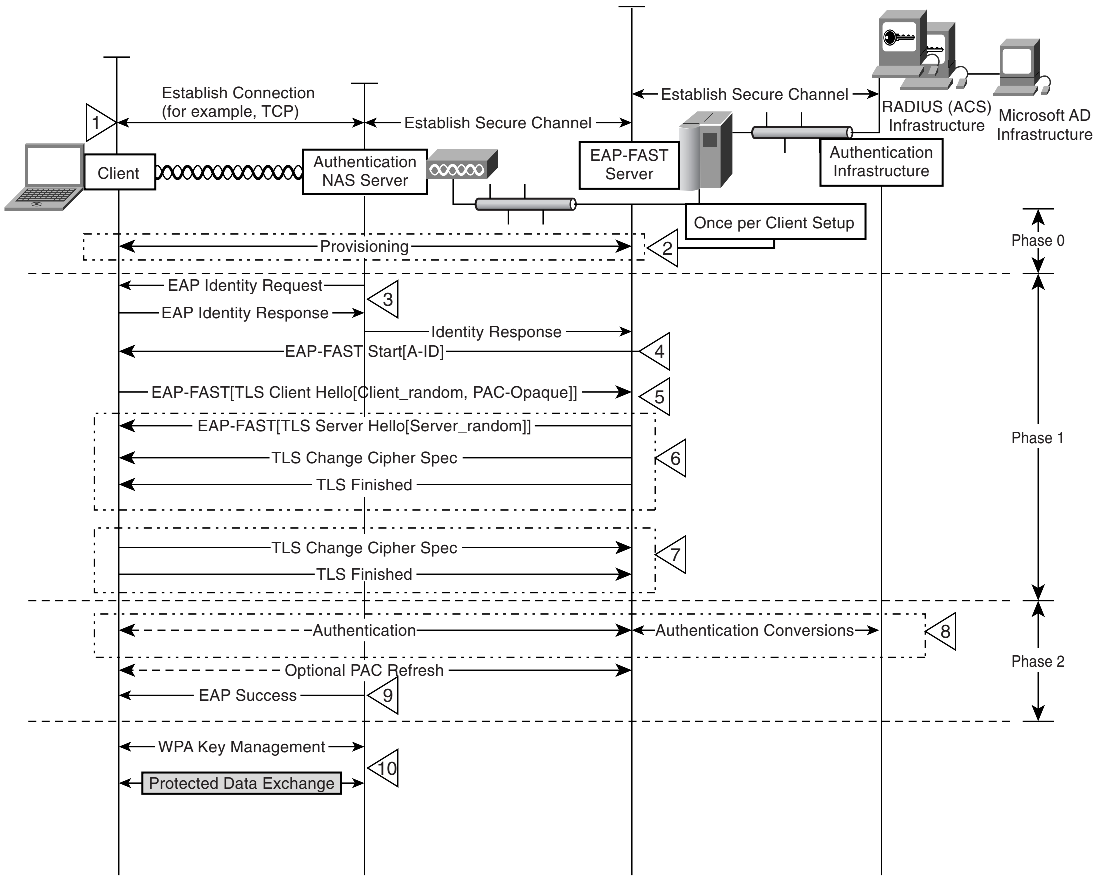


Table 7-4 *Salient Fields in EAP-FAST*

Name	Description	Relevance
A-ID	Authority identifier. This field would be in the EAP-FAST start frame.	A unique name identifying the authentication server. Will be used by the client/peer to index into the PAC and other context information.
I-ID	Initiator identifier.	A unique name identifying the peer/client.
CRED-LIFETIME	Expiration time of the credential.	This field will be in the PAC key info and used to validate a PAC key set.

Figure 7-21 *EAP-FAST Choreography Overview*



Step 1 Step 1, of course, is to have connectivity between the client/peer and AP, in addition to secure connections between the AP, EAP-FAST server, and authentication server.

Step 2 To bootstrap a secure channel, the EAP phase 0 provisioning needs to be performed. This is done once per client setup. This phase itself is an EAP-TLS exchange, with the Diffie-Hellman key exchange and fields embedded in the TLS choreography. At the end of phase 0, the PAC between the peer/client and the authentication server is established.

Step 3 This is similar to the EAP identity exchange.

Step 4 This is the EAP-FAST start message, which includes the authenticator ID.

Steps 5, 6, and 7 TLS exchanges over EAP-FAST to authenticate the peer and the server. The client sends the PAC-opaque to the server in Step 5.

Step 8 Step 8, the inner authentication method, is where the actual authentication happens. The message exchange is implemented via EAP-TLV over EAP-FAST between the peer and the EAP server and most probably RADIUS between the EAP server and the authentication server. It is also possible that the same software in one computer performs both server functions. The phase 2 inner authentication method over EAP-TLV can be EAP-SIM, EAP-OTP, EAP-GTC, or MSCHAPv2.

NOTE One of the built-in features in EAP-FAST is the PAC refresh, which can be done after successful authentication, at the end of Step 8. This functionality adds the secure update of the PAC as part of the EAP-FAST message exchange and infrastructure, thus making maintenance easier and more secure.

Step 9 This is the mandatory EAP success message required by EAP.

Step 10 You can now use the key materials and contexts established by the three phases to use WPA methods to exchange information, thereby achieving confidentiality and integrity.

Summary

This chapter examined the authentication methods: EAP, PEAP, LEAP, and the newer, emerging paradigm EAP-FAST. The chapter also dived into basic details about port-based access control: the 802.1x. As you can see, the various solutions are at a different maturity in terms of standardization and implementation. This scheme of things will probably continue for this year, and by 2005, this area will be more stabilized. The following chapter looks at the 802.11i and the wireless security blueprint.