

# WLAN Security

## 5. 802.1x and EAP Authentication

(C) Herbert Haas 2006/4/1

### Content

In this chapter a detailed overview about today's WLAN security problems and solutions are presented.

This subchapter provides an overview of 802.1x authentication and various EAP protocols.

### Objective

After completing this chapter the following tasks could be solved:

- Emphasize the basic vulnerabilities of WLAN
- Explain why WEP is insecure and give a mathematical example
- Compare WEP and TKIP/MIC
- Highlight the design flaws of the WLAN standard authentication
- Explain the design idea of 802.1x
- Compare EAP-TLS, LEAP, PEAP, EAP-TTLS, EAP-FAST with each other and emphasize important security features
- Explain the design concept of WPA and WPA2
- Implement a reliable 802.1x infrastructure over a WAN connection
- List important issues to be considered when choosing a VPN design
- Explain PSPF

# 802.1x Authentication – Intro



- **Port-based network access control method utilizing IETF's Extensible Authentication Protocol (EAP)**
  - ♦ Supports **mutual** authentication between client and AP
- **Dynamic WEP/TKIP key distribution and refresh**
  - ♦ Only for unicast traffic
    - Each client has its own key—as long as AP has enough key slots
    - Session lifetime
  - ♦ But static and shared broadcast key
    - Either pre-configured or automatically assigned after authentication
- **Centralized user credential management via RADIUS**
  - ♦ Various client credentials supported
- **(Fast) L2 roaming support (*possible*)**

The IEEE is working on a supplement to the 802.1d standard which will define the changes necessary to the operation of a MAC layer bridge in order to provide **port-based network access control** capability. This standard is known as 802.1x and has been adopted by the 802.11i working group.

802.1x provides port-based access control, that is, a special authentication mechanism is used to switch a bridge port or the AP from an unauthorized state into an authorized state. Only the latter state allows traffic other than 802.1x traffic.

Using 802.1x, a wireless client that associates with an AP cannot gain access to the network until the user performs a network logon or provides other strong credentials. Practically, when the user enters a username and password into a network logon dialog box or its equivalent, the client and an authentication server, a RADIUS server, perform a **mutual authentication**. Additionally, the RADIUS-based authentication server (AS) allows **centralized user credential management**.

Note that the AP acts as pass-through device, while the actual authentication process is performed by the authentication server. The authentication server and client then derive a client-specific WEP/TKIP key to be used by the client for the current logon session. User passwords and session keys are never transmitted in the clear, over the wireless link.

The whole authentication process is conducted by the **Extensible Authentication Protocol (EAP)** which has been defined in RFC 2284 as PPP extension. Note that EAP is only a meta-authentication protocol. EAP initiates the process and carries the actual authentication protocol, for example the Transport Layer Security (TLS) protocol and others. Most of them provide a session identifier and therefore provide seamless handover between access points, without re-authentication need.

Note that 802.1x can only negotiate per-user session keys for unicast transmission. A single **static broadcast key** must also be configured on an access point for 802.1x clients to receive broadcast and multicast messages. This is typically performed automatically.

Reauthentication can be easily realized, because each AP can ask the central AS whether the client is already authenticated. This principle supports fast roaming (even better, if there is a caching instance in-between).

**Wi-Fi's WPA** also requires 802.1x as authentication method.

# What is EAP?



- **Extensible: allows to develop and deploy new authentication protocols easily**
  - ◆ No SW update on authenticator (AP) needed
  - ◆ Only supplicant and AS server need to be updated
- **See RFC 2284**

TLS	MD5	AKA/SIM	TTLS	PEAP	FAST	LEAP
<b>EAP</b>						
802.1x "EAPoL" or "EAPoW"					RADIUS	
PPP		802.3		802.11		UDP
						IP
						802.3

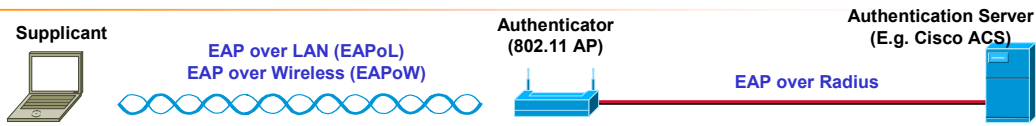
802.1x relies on EAP as underlying authentication protocol carrier. EAP is **extensible**, as it allows to develop and deploy new authentication protocols easily without changing the AP software. That is, EAP can be imagined as a container for authentication schemes.

The picture above shows the layers involved. EAP itself is either carried by a layer-2 protocol such as 802.3 ("EAP over LAN", **EAPoL**) or 802.11 ("EAP over Wireless", **EAPoW**), or by RADIUS ("**EAP over RADIUS**").

In order to be carried over RADIUS, the EAP information is decomposed into information elements and additionally, new Attribute Value Pairs (AVPs) had to be defined ("eap-radius").

See RFC 2284 for further details.

# 802.1x – Protocol Layers



EAP's Authentication Method			
EAP			
802.1x		802.1x	RADIUS
			UDP/IP
802.11		802.11	802.3

- **Authenticator (AP) blocks access until client is authenticated**
  - ◆ Only accepts Ethertype 0x888E (EAPoL)
- **802.1x frames are sent to multicast DA = 01-80-C2-00-00-03**
- **Authenticator translates 802.1x to UDP/IP**

Each 802.1x-based authentication consists of **three participants**:

1. The client, who is called "**Supplicant**"
2. The "**Authenticator**", which is actually the AP
3. An "**Authentication Server**" which must support eap-radius.

Both the Supplicant and the Authentication Server are authenticated to each other but this handshake is intercepted by the Authenticator, which forwards these messages to the endpoints.

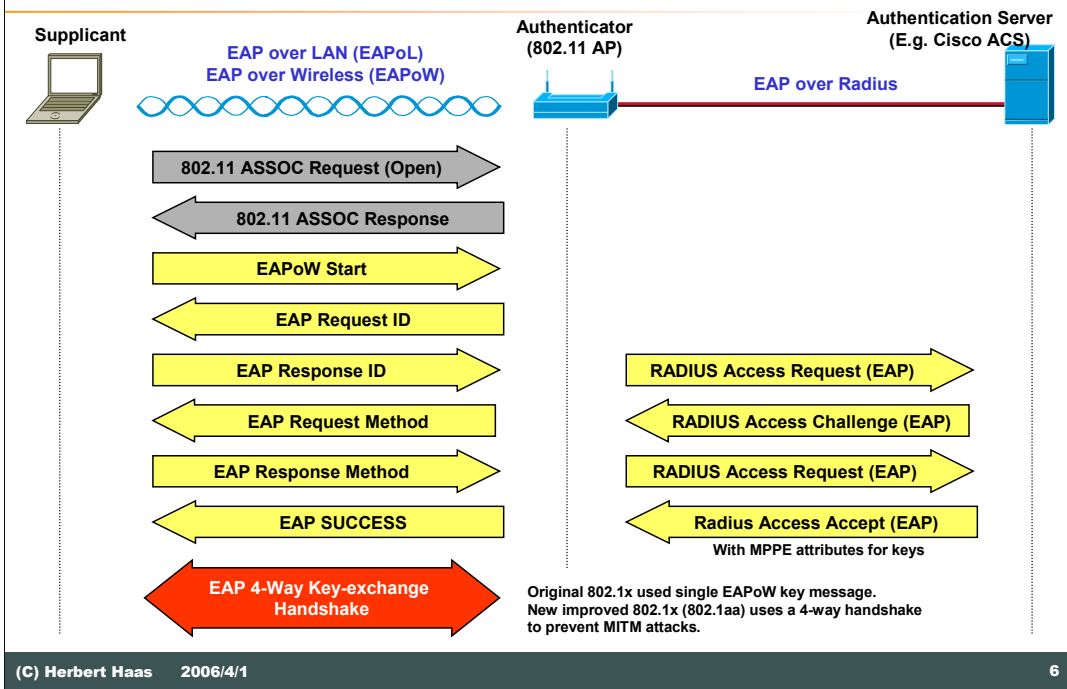
Of course also the Authenticator must be authenticated. This is typically done by a **shared secret** between the Authenticator and the Authentication Server.

**Note:** The Authenticator is basically an 802.1x-to-UDP "bridge".

When an 802.1X-capable host starts up, it will initiate the authentication phase by sending the EAPoL-Start 802.1x protocol data unit (PDU) to the reserved IEEE multicast MAC address (01-80-C2-00-00-03) with the Ethernet type or length set to **0x888E**.



# 802.1x – EAP Protocol



EAP provides an envelope that can carry many **different kinds of authentication types**: challenge/response, one time passwords (OTPs), SecurID tokens, digital certificates, etc. What exactly happens between "EAP Start" and "EAP Success" depends upon the type of authentication being used.

The original 802.1X standard used a single EAPoW Key message for this purpose, but the **new improved 802.1x** (called 802.1aa) uses a **four-way handshake to prevent man-in-the-middle attacks** that might otherwise compromise these keys. After both ends—the client and the AP—of the wireless association have session keys, data sent over the air can be encrypted to prevent eavesdropping.

One of the most important benefits will be felt by users who are **roaming** within an organization's wireless LAN and require a seamless connection. If they are asked to authenticate themselves each time they pass from one conference room to another, they will want to give up security in favor of convenience.

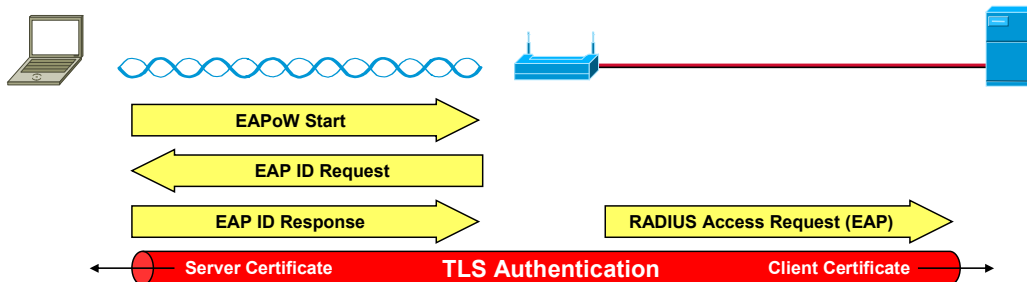
Using the connection re-establishment mechanism provided by the TLS handshake users can have one seamless connection while roaming between different APs connected to the same backend server. If the **session ID** is still valid, the wireless client and server can share previously negotiated secrets to establish a new handshake and keep the connection alive.

Additionally, secure **session timeouts** trigger re-authentication and new WEP (TKIP) keys.

# 802.1x – EAP-TLS (1)



- First secure 802.1x realization, EAP method 13 (RFC 2716)
- Relies on Transport Layer Security (TLS)
  - Successor of SSL version 3.0, adopted by IETF
  - Both clients and AS authenticated via certificates
  - *Only TLS authentication and tunnel establishment procedure (tunnel not used)*
  - TLS also used to derive link-layer key between endpoints
- Problems:
  - Client identity is not protected
  - No fast session reconnection
  - Need for PKI (practical: certificate stored in token card or similar)
- Prerequisite for WPA certification
  - Until May 2005 the only required EAP method for WPA



(C) Herbert Haas 2006/4/1

7

The **TLS Working Group** was established in 1996 to standardize a "transport layer" security protocol. The working group began with SSL version 3.0, and in 1999, RFC 2246, the "TLS Protocol Version 1.0" was published as a Proposed Standard. The working group has also published RFC 2712, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)" as a Proposed Standard, and two RFCs on the use of TLS with HTTP.

EAP-TLS was the first **most-widely implemented** 802.1x method for WLANs. EAP-TLS supports **session expiration** and 802.1x re-authentication by using the RADIUS session timeout option (RADIUS Internet Engineering Task Force option 27). To avoid IV reuse (IV collisions), the base **WEP key is rotated** before the IV space is exhausted.

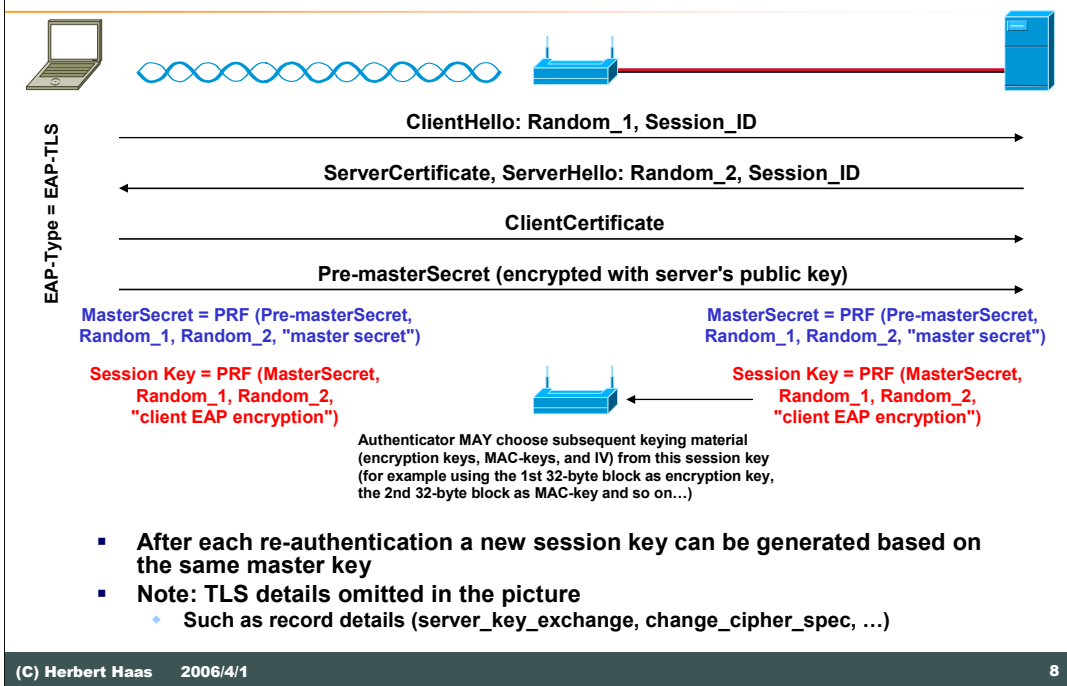
However, **several problems lead to a seldom use** of EAP-TLS:

- Every client needs a certificate. This is only affordable if a PKI is available providing a CA for certificate management, revocation and so on.
- In the three messages of the EAP starting sequence, the user-ID is revealed. This is considered privacy-critical today.
- Fast session reconnection (for VoIP) is not possible.

**Remember:** A certificate is a cryptographically signed structure, that guarantees the association between at least one *identifier* and a *public key*.

**Note:** The name in the client certificate must be the same username as in the AS user database. This is one important reason to choose a private Root-CA, besides the advantage of more control.

## 802.1x – EAP-TLS (2)



The Session\_ID can be used for fast re-authentication purposes.

As part of the TLS handshake between the server and the client, the client generates a **pre-master secret** and encrypts it with the server's public key. Then this pre-master secret is sent to the AS. Another option would be to use Diffie-Hellman exchange to derive the pre-master secret.

The pre-master secret, server and client random values, and "master secret" string value are used to generate a **master secret** per session. A Pseudo Random Function (PRF) is used again along with master secret, client and server random values, and "client EAP encryption" string value to generate the 128-bit **session keys**, Message Authentication Code (MAC) keys and initialization values (for block ciphers only).

Note that both the client and the AS independently derive the session keys. However, the length of the session key is determined by the authenticator (the AP) and is sent in the EAPoL key message at the end of the EAP authentication to the client.

A TLS session is governed by a security context, which consists of session identifier, peer certificate, compression method, cipher spec for the session key, MAC algorithm parameters, and the shared master secret.

TLS sessions expire after some time and the AS can be notified via RADIUS.

*EAP-TLS is natively supported in MAC OS 10.3 and above, Windows 2000 SP4, Windows XP, Windows Mobile 2003 and above, and Windows CE 4.2*

# 802.1x – LEAP



- Cisco's lightweight implementation
- Fast Secure Roaming (< 150 ms)
- Challenge-response based on shared secrets
  - ♦ Implemented similar as MS-CHAPv2 (two stage MD4 hashing of passwords)
- Can utilize existing Windows NT Domain Services authentication databases as well as Windows 2000 Active Directory databases
  - ♦ No support for LDAP and NIS
- Drivers for Windows 95, 98, Me, 2000, NT and XP and uses the Windows logon as the Cisco LEAP logon
- Also Linux and Mac support
- Vulnerable to dictionary attacks
  - ♦ Secure if strong passwords are enforced (10 chars at minimum)

Cisco's **Lightweight EAP (LEAP)** implementation is widely deployed because of its simplicity as it is based on **shared user secrets**. Furthermore, only LEAP supports **fast secure roaming**, necessary if low-delay applications are used (e. g. VoIP).

Cisco has developed drivers for most versions of Microsoft Windows (Windows 95, 98, Me, 2000, NT and XP) and uses the **Windows logon** as the Cisco LEAP logon.

A software shim in the Windows logon allows the username and password information to be passed to the Cisco Aironet client driver. The driver will convert the password into a Windows NT key and hand the username and Windows NT key to the Cisco NIC. The NIC executes 802.1x transactions with the AP and the authentication, authorization, and accounting (AAA) server.

**Note:** Neither the password nor the password hash is ever sent across the wireless medium.

Additionally, any Open Database Connectivity (ODBC) that uses MS-CHAP passwords can also be used with LEAP.

**Note:** If an AS is used for both Cisco LEAP and MAC authentication, the MAC address should use a different strong password for the required MS-CHAP/CHAP field. If not, an eavesdropper can spoof a valid MAC address and use it as a username and password combination for Cisco LEAP authentication.

**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.

To avoid IV reuse (IV collisions), LEAP rotates the base WEP key before the IV space is exhausted.

**Note:** LEAP is only as strong as the passwords used. Therefore it is vulnerable to dictionary attacks. At least 10-character passwords should be used.

**BTW:** Implementation details of LEAPv1: ChallengeLEN=8, RESPONSE\_LEN=24, KEY\_LEN=16 [BYTES]

# LEAP / MSCHAPv2 Flaws



- **AS sends 8 byte challenge**
- **Client encrypts challenge 3 times using NT hash of the password as DES seed (=key)**
  - ♦ DES requires a 7 byte seed value in this algorithm
  - ♦ So client splits 16 byte NT hash into three portions:
    - Seed1 = B1 .. B7
    - Seed2 = B8 .. B14
    - Seed3 = B15, B16, 0x00, 0x00, 0x00, 0x00, 0x00
- **Flaw: third DES output is cryptographically weak, leaving only  $2^{16}$  possible permutations**
- **After B15 and B16 are known, we can significantly reduce the number of potential matches in our dictionary file, using the known 2 bytes of the user's hash as a keying mechanism**

8 Byte challenge is encrypted 3 times, using Seed3 for the third DES encryption. Since the attacker knows the challenge and the encrypted response, a simple brute force attack quickly recovers seed3. Now the search duration in the attacker's dictionary file can be significantly reduced. Assuming that this dictionary file has been prepared such that it already contains the NT hashes of each password, the lookup algorithm must only look for hashes for which bytes 15 and 16 matches the recovered seed3.

# Asleap



- **Offline attack on LEAP**
- **Principle:**
  - ◆ LEAP performs unencrypted MSCHAPv2 (challenge-handshake)
  - ◆ Asleap captures challenge and encrypted reply and performs an offline dictionary attack
- **Written by Joshua Wright**
- **<http://asleap.sourceforge.net/>**
- **Also see Leapcrack**

```
root@cyanocorax: tools/asleap-1.0
File Edit View Terminal Go Help
asleap 1.0 - actively recover LEAP passwords. <jwright@hasborg.com>
Using the passive attack method.

Captured LEAP challenge:
0802 d500 00d0 59c8 6119 0040 9655 2d21 .....Y.a..@.U-!
0040 9655 2d21 006d aaaa 0300 0000 888e ..@.U-!..m.....
0100 0014 0122 0014 1101 0008 7e46 733d ...$.".$......Fs=
63a5 fabf 6265 7374 .....c...best

Captured LEAP response:
0801 d500 0040 9655 2d21 00d0 59c8 6119 .....@.U-!..Y.a.
0040 9655 2d21 b021 aaaa 0300 00f8 888e ..@.U-!..l.....
0100 0024 0222 0024 1101 0018 d51b 8d53 ...$.".$......S
c087 9888 fdee 7e85 0a08 add4 626b d61b .....bk..
d66e 53a7 6265 7374 .....n.S.best

Captured LEAP auth success:
0802 d500 000c 3043 a907 0007 50ca f417 .....0C...P...
0007 50ca f417 5067 aaaa 0300 0000 888e ..P...Pg.....
0100 0004 0313 0004 .....

Captured LEAP exchange information:
username: best
challenge: 7e46733d63a5fabf
response: d51b8d53c0879888fdee7e850a08add4626bd01bd6e53a7
Attempting to recover last 2 of hash.
hash bytes: 9537
Starting dictionary lookups.
NT hash: 0cb6948805f797bf2a82807973b89537
password: test
[root@cyanocorax asleap-1.0]#
```

Example: Asleap, cracking password "test"

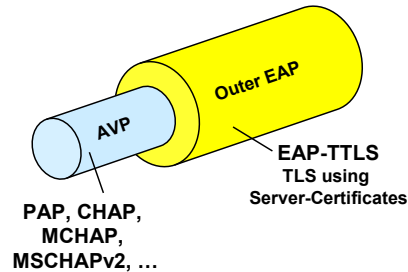
good policy should also require a password length of at least 12 characters, including numbers, mixed case, and punctuation. It should also include a requirement that passwords be based on neither words found in any dictionary nor any variant of the username. There are cracking dictionaries for hundreds of languages and commonly used words, such as names of places, people, and movies. Usually the only way to enforce strong passwords is with tools that enforce passwords at creation time. Users are good at choosing easy-to-remember passwords and tend to ignore unenforced rules. It is a good idea to run regular, automated password cracking on your organization's passwords and warn users or disable accounts with bad passwords. Your organizational environment determines what strength of password enforcement and frequency of password changes is acceptable to your user community.

# 802.1x – EAP-TTLS



- Created by Funk and Certicom (Internet draft)
- EAP method 21
- Widely implemented, also Linux support; but no Cisco support
- Supports ANY inner authentication method
  - ◆ Any EAP method
  - ◆ As well as older methods such as CHAP, PAP, MS-CHAP and MS-CHAPv2

Basic Idea:



EAP-TTLS was developed by Funk Software and Certicom, and was first supported by Agere Systems, Proxim, and Avaya. Today EAP-TTLS is being considered by the IETF as a new standard.

The structure of **Tunnelled TLS (TTLS)** and PEAP are **quite similar**. Both are two-stage protocols that establish security in stage one and then exchange authentication in stage two.

Stage one of both protocols establishes a TLS tunnel and authenticates the authentication server to the client with a **certificate**. Once that secure channel has been established, client authentication credentials are exchanged in the second stage.

Other than PEAP, EAP-TTLS supports **any authentication method**, not only EAP-methods. Therefore, there is no inner EAP session but **RADIUS-like AVPs** are used to carry the authentication data.

EAP-TTLS often uses PAP (also with Linux).

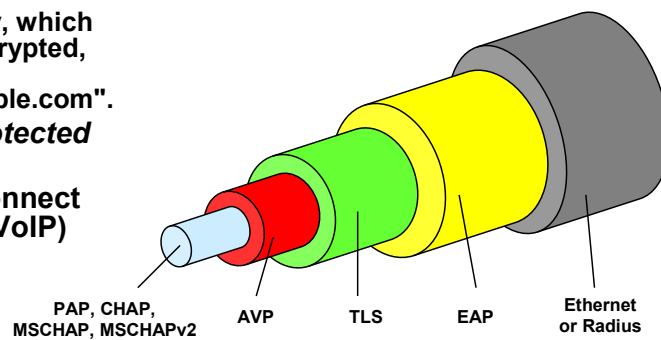
As with PEAP, **user identity information is protected**.

# 802.1x – EAP-TTLS



- Radius-like AVPs between client and Server
- Client certificate not required but user has two identities:
  1. A anonymous identity such as "anonymous@example.com" and
  2. The real identity, which is only sent encrypted, such as "user342@example.com".
- *Client identity protected by TLS*
- Fast session reconnect (but too slow for VoIP)

Detailed:



## 802.1x – Other EAP Choices



- **More than 44 EAP types already defined**
  - ♦ EAP-AKA: username and password
  - ♦ EAP-MD5: No dynamic WEP keys, no mutual authentication, dictionary attacks possible (EAP method 4)
  - ♦ EAP-GTC: Generic Token Card (EAP method 6), no mutual authentication
  - ♦ PEAP-GTC: Cisco's PEAP method
  - ♦ EAP-SIM: Used for SIM-card based devices (3GPP, also known as EAP-GSM)
  - ♦ EAP-SRP: Secure Remote Password
  - ♦ ...
- **EAP-FAST: Successor of LEAP**
  - ♦ See dedicated section
- **PEAP-EAP-TLS**
  - ♦ Another Microsoft solution similar as EAP-TLS

There are other EAP methods which are currently not so important in the 802.11 WLAN world.

**EAP-AKA** works similar as LEAP. AKA stands for Authentication and Key Agreement. It is also used with HTTP Authentication and GSM. See *draft-arkko-pppext-eap-aka-12.txt* for details.

**EAP-MD5** does not support mutual authentication and is not strong enough, also some vendors use it with WLAN devices.

**EAP-GTC** is typically only used as inner EAP-method of PEAP. In this case it is often called "PEAP-GTC".

**EAP-SIM** is used by 3GPP applications (GSM and UMTS). SIM stands for Subscriber Identity Module.

**EAP-SRP** (Secure Remote Password) is a method used by some vendors, mainly Orinoco.

**WPA-Note:** EAP-MD5, EAP-GTC, EAP-OTP, and EAP-MSCHAPV2 cannot be used alone with WPA. They can only be used as inner authentication algorithms with EAP-PEAP and EAP-TTLS.

Microsoft supports another form of PEAPv0 (which Microsoft calls **PEAP-EAP-TLS**) that Cisco and other third-party server and client software don't support.

PEAP-EAP-TLS does require a client-side digital certificate located on the client's hard drive or a more secure smartcard. PEAP-EAP-TLS is very similar in operation to the original EAP-TLS but provides slightly more protection due to the fact that portions of the client certificate that are unencrypted in EAP-TLS are encrypted in PEAP-EAP-TLS.

Since few third-party clients and servers support PEAP-EAP-TLS, users should probably avoid it unless they only intend to use Microsoft desktop clients and servers.

# EAP Types Overview



- 1–6 Assigned by RFC
  - 1Identity
  - 2Notification
  - 3Nak (response only)
  - 4MD5-Challenge
  - 5One-Time Password (OTP)
  - 6Generic Token Card (GTC)
- 7-8 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
- 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 SentiNET
- 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

This list is just for reference.

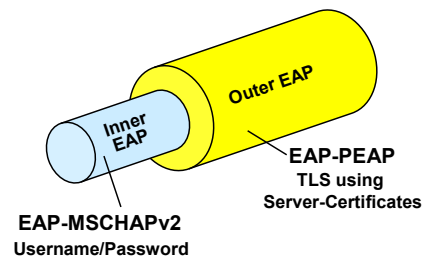
**PEAP**

# 802.1x using PEAP



- **Created by Cisco and Microsoft**
  - ◆ Similar to EAP-TTLS
- **Open standard**
  - ◆ EAP method 25
- **Since third EAP message is always in clear**
  - ◆ Client may send a routing realm instead of the user identity to protect the user identity

Basic Idea:



(C) Herbert Haas 2006/4/1

17

**Protected EAP (PEAP)** has been developed by **Cisco and Microsoft** and is only available on the newest Microsoft platforms (XP).

PEAP is a **two-stage protocol** that establish a secure TLS tunnel which carries an **inner EAP session**.

PEAP only supports EAP-type authentication. Microsoft proposes MS-CHAPv2, while Cisco prefers Generic Token Cards (EAP-GTC). Cisco differentiates "v0" and "v1" while Microsoft only knows "PEAP", which means PEAPv0 and only supports MSCHAPv2. Cisco's implementation "v0" also supports EAP-SIM, while "v1" also supports EAP-GTC.

The main advantage of PEAP is that **client certificates are not necessary**.

Support for MS-DB but no support for LDAP-DB.

The PEAP result is the so-called **Compound Session Key (CSK)** which 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:

MSK: Key derived between the peer and the EAP server and exported to the authenticator.

EMSK: 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.

# Version Overview



- **PEAPv0**
  - ◆ Supported since Windows XP SP1
  - ◆ Microsoft proposes MS-CHAPv2
    - Inner EAP method 29
- **PEAPv1**
  - ◆ Cisco's proposal: EAP-GTC
    - Inner EAP method 6
- **PEAPv2**
  - ◆ Latest draft
  - ◆ Security updates and more features
    - Various cipher-suites supported
    - **MITM protection through "crypto-binding"**

The PEAP result is the so-called **Compound Session Key (CSK)** which 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:

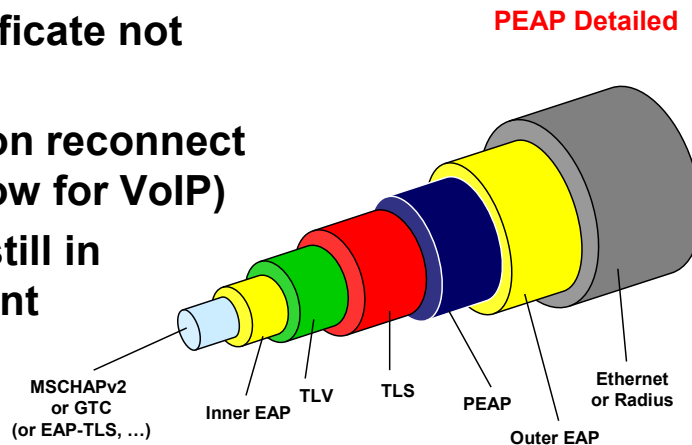
MSK: Key derived between the peer and the EAP server and exported to the authenticator.

EMSK: 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.

# PEAP as Pipe Model



- **Only supports EAP-type authentication**
- **Client certificate not required**
- **Fast session reconnect (but too slow for VoIP)**
- **Version 2 still in development**



## Security Claims of PEAPv2

**Intended use:** Wireless or Wired networks, and over the Internet, where physical security cannot be assumed.

**Auth. mechanism:** Use arbitrary EAP and TLS authentication mechanisms for authentication of the client and server.

**Ciphersuite negotiation:** Yes.

**Mutual authentication:** Yes. Depends on the type of EAP method used within the tunnel and the type of authentication used within TLS.

**Integrity protection:** Yes

**Replay protection:** Yes

**Confidentiality:** Yes

**Key derivation:** Yes

**Key strength:** Variable

**Dictionary attack prot:** Not susceptible.

**Fast reconnect:** Yes

**Crypt. binding:** Yes.

**Acknowledged S/F:** Yes

**Session independence:** Yes.

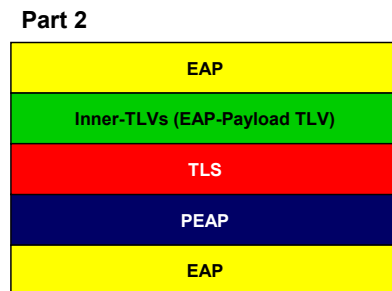
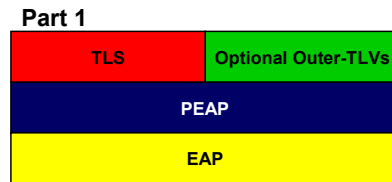
**Fragmentation:** Yes

**State Synchronization:** Yes [80211Req]

# PEAPv2 Layers



- In PEAPv2 Part 1
  - ◆ Outer-TLVs are used to help establishing the TLS tunnel, but no Inner-TLVs are used
- In PEAPv2 Part 2
  - ◆ TLS records may encapsulate zero or more Inner-TLVs, but no Outer-TLVs
  - ◆ EAP packets used within tunneled EAP authentication methods are carried within Inner-TLVs



The TLS v1.0 mandatory-to-implement ciphersuite `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA` must be supported

For light-weight devices also other TLS cipher suites supported

PEAPv2 client and servers SHOULD support

`TLS_RSA_WITH_3DES_EDE_CBC_SHA`

`TLS_RSA_WITH_RC4_128_MD5`

`TLS_RSA_WITH_RC4_128_SHA`

`TLS_RSA_WITH_AES_128_CBC_SHA`

## PEAPv2: Provisioning of Credentials



- **Provisioning inside a server-authenticated TLS tunnel**
- **Provisioning inside a server-unauthenticated TLS tunnel**
  - ◆ **If TLS tunnel cannot be validated by client (lacking required credentials) the client instead may rely on inner EAP method**
  - ◆ **Although this reduces deployment costs, MITM attacks are possible !**
  - ◆ **An implementation is therefore optional and not recommended**

Unfortunately many people use PEAP inside a "server-unauthenticated TLS tunnel" which is (unfortunately) a supported method – but this actually conflicts with the initial idea of a secure-tunnel authentication!

Therefore always install appropriate root certificates!



- **Also other than certificate-based cipher-suites are supported**
  - ◆ E. g. DH-based
- **If certificates are sent by the server**
  - ◆ The client only verifies whether the server possesses the corresponding private key
  - ◆ The client does not need to validate via the trust anchor (CA)

If the validation of the server certificate fails (because of failing private key validation or invalid certificate parameters) then the "provisioning inside a server-unauthenticated TLS tunnel"-mode must not be entered.

## PEAPv2 – MITM Protection

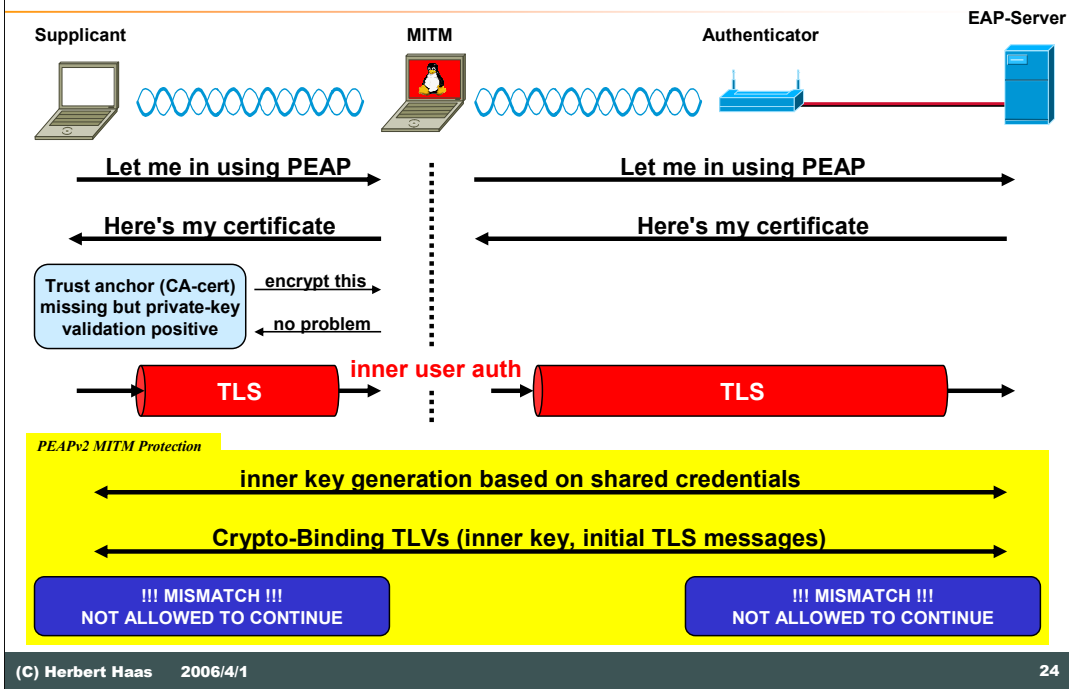


- A **sequence** of zero or more inner EAP authentication methods can be negotiated
- **Crypto-Binding TLVs** must be sent in the PEAP success/failure (Result TLV) messages
  - ◆ In a sequence, also after each EAP-method a Crypto-Binding TLV must be sent by both parties
  - ◆ The server should not reveal any sensitive data to the client until after the Crypto-Binding TLV has been properly verified !!!

Note that with every EAP method, there must be a final EAP success/failure indication sent in clear – to inform the authenticator.

An early Cisco solution to the MITM problem with pre-PEAPv2 versions is to enforce the client to choose a PEAP trust anchor. That is the client must select a root certificate issuer from a list. If the certificate offered by the server cannot be validated via the pre-selected trust anchor, the authentication process stops. Unfortunately, also "any" can be selected.

# PEAP: Man-In-The-Middle Attack



A man-in-the-middle can spoof the client to authenticate to it instead of the real EAP server; and forward the authentication to the real server over a protected tunnel. Since the attacker has access to the keys derived from the tunnel, it can gain access to the network.

PEAP version 2 prevents this attack by using the keys generated by the inner EAP method in the crypto-binding exchange described in protected termination section. This attack is not prevented if the inner EAP method does not generate keys or if the keys generated by the inner EAP method can be compromised. Hence, in cases where the inner EAP method does not generate keys, the recommended solution is to always deploy authentication methods protected by PEAPv2.

# Crypto-Binding TLVs



- **PEAPv2 derives keys by combining keys from TLS and the inner EAP methods**
- **The Crypto-Binding TLV calculation includes**
  - ◆ **The first two Outer-TLVs messages sent by both peer and EAP-server**
    - (used for TLS tunnel establishment)
  - ◆ **The EAP-Type (= set to PEAP) sent in the first two messages by both peer and EAP-server**

Outer-TLVs SHOULD NOT be included in other PEAP packets since there is no mechanism to detect modification.

For *subsequent* packets (after the first two) the EAP Type in the clear could be modified and will likely result in failure, hence it is not included in the Crypto-Binding calculation.



- **Theoretically possible if the attacker**
  - ◆ **Can modify unprotected fields in the PEAP packet such as the EAP protocol or PEAP version number**
  - ◆ **Modify protected fields in a packet to cause decode errors**

## PEAPv2 – Other Features



- **Fast session resumption**
  - ♦ **Using the "sessionID" of the TLS protocol and the Server-Identifier TLV in PEAP**
    - **Server may send a Server-Identifier TLV to give client a hint which sessionID should be used (protected by MAC)**
  - ♦ **If too much time elapsed since previous authentication, the server will not allow the continuation**
  - ♦ **The inner authentication may or may not be skipped !!!**
- **TLS compression must be supported**

PEAPv2 "fast reconnect" is desirable in applications such as wireless roaming, since it minimizes interruptions in connectivity. It is also desirable when the "inner" EAP mechanism used is such that it requires user interaction. The user should not be required to re-authenticate herself, using biometrics, token cards or similar, every time the radio connectivity is handed over between access points in wireless environments.

Since PEAPv2 Part 1 may not provide client authentication, establishment of a TLS session (and an entry in the TLS session cache) does not by itself provide an indication of the peer's authenticity. Implementations that do not remove TLS session cache entries after a failed PEAPv2 Part 2 authentication or failed protected termination **MUST** use other means than successful TLS resumption as the indicator of whether the client is authenticated or not. TLS resumption **MUST** only be enabled if the implementation supports TLS session cache removal !!!

If an EAP server implementing PEAPv2 removes TLS session cache entries of peers failing PEAPv2 Part 2 authentication, then it **MAY** skip the PEAPv2 Part 2 conversation entirely after a successful session resumption, successfully terminating the PEAPv2 conversation

# PEAPv2 Fragmentation



- **A single TLS message may consist of multiple TLS records**
  - ♦ A single TLS record may be up to 16384 bytes in length
  - ♦ A TLS certificate message may in principle be as long as 16 MByte
- **Fragmentation needed**
  - ♦ RADIUS cannot handle such long messages
  - ♦ Multilink PPP (MRRU LCP) method supported on Ethernet/802.3
    - But there's no PPP in 802.11 which could negotiate that
  - ♦ PEAPv2 own fragmentation support defined
    - DoS attacks (reassemble lockup) can be mitigated to set a maximum size for one group of TLV messages (e. g. 64 KB)

Fragmentation support is not that easy. Requires sequence numbers ACKs and NAKs (fortunately provided by EAP already), several flags such as (M)ore fragments, (S)tart and a length field.

# PEAPv2 Key Derivation



- **New keys are derived from TLS master secret to protect the conversation within the PEAPv2 tunnel**
  - ◆ Since normal TLS keys are used in the handshake they should not be used in a different context
- **Combines key material from TLS exchange with key material from inner key generating EAP methods**
  - ◆ To bind inner authentication mechanisms to TLS tunnel

The input for the cryptographic binding includes the following:

[a] The PEAPv2 tunnel key (TK) is calculated using the first 40 octets of the (secret) key material generated as described in the EAP-TLS algorithm ([RFC2716] Section 3.5). More explicitly, the TK is the first 40 octets of the PRF as defined in [RFC2716]:

PRF(master secret,"client EAP encryption", random)

Where random is the concatenation of client\_hello.random and server\_hello.random

[b] The first 32 octets of the MSK provided by each successful inner EAP method ;for each successful EAP method completed within the tunnel.

ISK1..ISK<sub>n</sub> are the MSK portion of the EAP keying material obtained from methods 1 to n. The ISK<sub>j</sub> shall be the first 32 octets of the generated MSK of the jth EAP method. If the MSK length is less than 32 octets, it shall be padded with 0x00's to ensure the MSK is 32 octets. Similarly, if no keying material is provided for the EAP method, then ISK<sub>j</sub> shall be set to zero (e.g. 32 octets of 0x00).

The PRF algorithm is based on PRF+ from IKEv2 shown below ("|" denotes concatenation)

K = Key, S = Seed, LEN = output length, represented as binary in a single octet.

PRF (K,S,LEN) = T1 | T2 | T3 | T4 | ... where:

T1 = HMAC-SHA1(K, S | LEN | 0x01)

T2 = HMAC-SHA1 (K, T1 | S | LEN | 0x02)

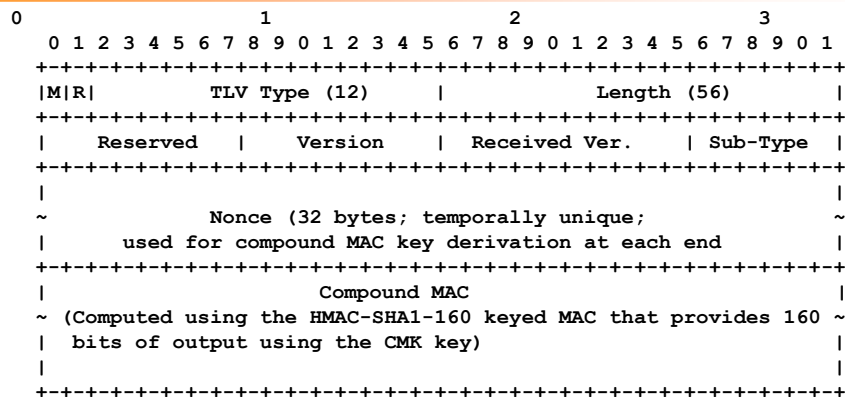
T3 = HMAC-SHA1 (K, T2 | S | LEN | 0x03)

T4 = HMAC-SHA1 (K, T3 | S | LEN | 0x04)

...

The intermediate combined key is generated after each successful EAP method inside the tunnel.

# Crypto-Binding TLV



- **The Crypto-Binding TLV is used prove that both peers participated in the sequence of authentications**
  - ◆ **That is, the TLS session and inner EAP methods that generate keys**

The Crypto-Binding TLV MUST be used to perform Cryptographic Binding after each successful EAP method in a sequence of EAP methods is complete in PEAPv2 part 2. The Crypto-Binding TLV can also be used during Protected Termination.

The Crypto-Binding TLV must have the version number received during the PEAP version negotiation. The receiver of the Crypto-Binding TLV must verify that the version in the Crypto-Binding TLV matches the version it sent during the PEAP version negotiation. If this check fails then the TLV is invalid.

The receiver of the Crypto-Binding TLV must verify that the subtype is not set to any value other than the ones allowed. If this check fails then the TLV is invalid.

This message format is used for the Binding Request (B1) and also the Binding Response. This uses TLV type CRYPTO\_BINDING\_TLV. PEAPv2 implementations MUST support this TLV and this TLV cannot be responded to with a NAK TLV.

### The MAC is computed over:

1. The entire Crypto-Binding TLV attribute with the MAC field zeroed out.
2. The EAP Type sent by the other party in the first PEAP message.
3. All the Outer-TLVs from the first PEAP message sent by EAP-server to peer. If a single PEAP message is fragmented into multiple PEAP packets; then the Outer-TLVs in all the fragments of that message MUST be included.
4. All the Outer-TLVs from the first PEAP message sent by the peer to the EAP server. If a single PEAP message is fragmented into multiple PEAP packets, then the Outer-TLVs in all the fragments of that message MUST be included.