Network Working Group
Request for Comments: 4746
Category: Informational

T. Clancy
LTS
W. Arbaugh
UMD

November 2006

Extensible Authentication Protocol (EAP) Password Authenticated Exchange

Status of This Memo

Copyright Notice

Copyright Notice

Abstract

This document defines an Extensible Authentication Protocol (EAP) method called EAP-PAX (Password Authenticated eXchange). This method is a lightweight shared-key authentication protocol with optional support for key provisioning, key management, identity protection, and authenticated data exchange.

Table of Contents

1. Introduction

EAP-PAX (Password Authenticated eXchange) is an Extensible Authentication Protocol (EAP) method [RFC3748] designed for authentication using a shared key. It makes use of two separate subprotocols, PAX_STD and PAX_SEC. PAX_STD is a simple, lightweight protocol for mutual authentication using a shared key, supporting Authenticated Data Exchange (ADE). PAX_SEC complements PAX_STD by providing support for shared-key provisioning and identity protection using a server-side public key.

The idea motivating EAP-PAX is a desire for device authentication bootstrapped by a simple Personal Identification Number (PIN). If a weak key is used or a expiration period has elapsed, the authentication server forces a key update. Rather than using a symmetric key exchange, the client and server perform a Diffie-Hellman key exchange, which provides forward secrecy.

Since implementing a Public Key Infrastructure (PKI) can be cumbersome, PAX_SEC defines multiple client security policies, selectable based on one's threat model. In the weakest mode, PAX_SEC allows the use of raw public keys completely eliminating the need for a PKI. In the strongest mode, PAX_SEC requires that EAP servers use certificates signed by a trusted Certification Authority (CA). In the weaker modes, during provisioning PAX_SEC is vulnerable to a man-in-the-middle dictionary attack. In the strongest mode, EAP-PAX is provably secure under the Random Oracle model.

EAP-PAX supports the generation of strong key material; mutual authentication; resistance to desynchronization, dictionary, and man-in-the-middle attacks; ciphersuite extensibility with protected negotiation; identity protection; and the authenticated exchange of data, useful for implementing channel binding. These features satisfy the EAP method requirements for wireless LANs [RFC4017], making EAP-PAX ideal for wireless environments such as IEEE 802.11 [IEEE.80211].

1.1. Language Requirements

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Terminology

This section describes the various variables and functions used in the EAP-PAX protocol. They will be referenced frequently in later sections.

Variables:

CID
: User-supplied client ID, specified as a Network Access Identifier (NAI) [RFC4282], restricted to 65535 octets

g
: public Diffie-Hellman generator, typically the integer 2 [RFC2631]

M
   128-bit random integer generated by the server

N
   128-bit random integer generated by the client

X
   256-bit random integer generated by the server

Y
   256-bit random integer generated by the client

Keys:

AK
   authentication key shared between the client and EAP server

AK'
   new authentication key generated during a key update

CertPK
   EAP server's certificate containing public key PK

CK
   Confirmation Key generated from the MK and used during
   authentication to prove knowledge of AK

EMSK
   Extended Master Session Key also generated from the MK and
   containing additional keying material

IV
   Initialization Vector used to seed ciphers; exported to the
   authenticator

MID
   Method ID used to construct the EAP Session ID and consequently
   name all the exported keys [IETF.KEY]

MK
   Master Key between the client and EAP server from which all other
   EAP method session keys are derived

MSK
   Master Session Key generated from the MK and exported by the EAP
   method to the authenticator

PK
EAP server's public key

Operations:

enc_X(Y)
encryption of message Y with public key X

MAC_X(Y)
keyed message authentication code computed over message Y with symmetric key X

PAX-KDF-W(X, Y, Z)
PAX Key Derivation Function computed using secret X, identifier Y, and seed Z, and producing W octets of output

||
string or binary data concatenation

2. Overview

The EAP framework [RFC3748] defines four basic steps that occur during the execution of an EAP conversation between client and server. The first phase, discovery, is handled by the underlying link-layer protocol. The authentication phase is defined here. The key distribution and secure association phases are handled differently depending on the underlying protocol, and are not discussed in this document.

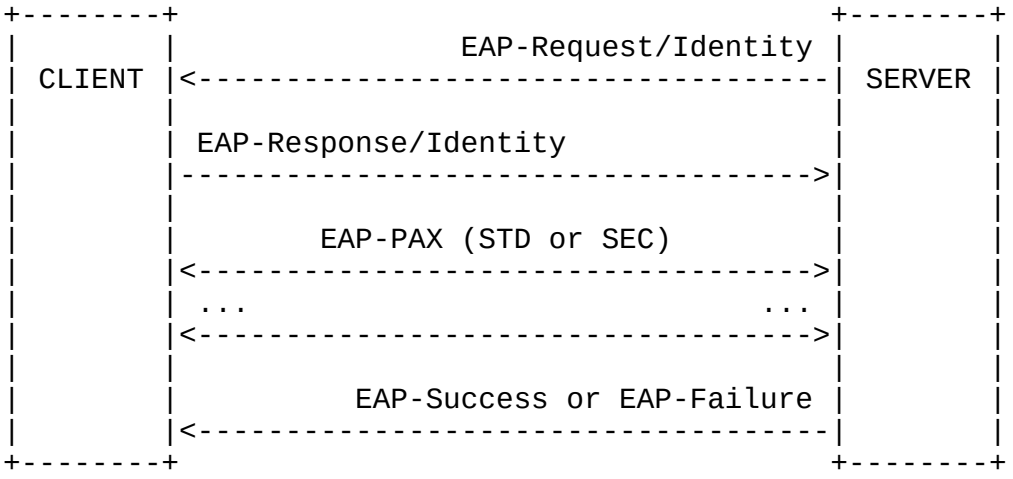

Figure 1: EAP-PAX Packet Exchanges

There are two distinct subprotocols that can be executed. The first, PAX_STD, is used during typical authentications. The second, PAX_SEC, provides more secure features such as key provisioning and identity protection.

PAX_STD and PAX_SEC have two modes of operation. When an AK update is being performed, the client and server exchange Diffie-Hellman exponents g^X and g^Y, which are computed modulo prime P or over an elliptic curve multiplicative group. When no update is being performed, and only session keys are being derived, X and Y are exchanged. Using Diffie-Hellman during the key update provides forward secrecy, and secure key derivation when a weak provisioned key is used.

The main deployment difference between PAX_STD and PAX_SEC is that PAX_SEC requires a server-side public key. More specifically, that means a private key known only to the server with corresponding public key being transmitted to the client during each PAX_SEC session. For every authentication, the client is required to compute computationally intensive public-key operations. PAX_STD, on the other hand, uses purely symmetric operations, other than a possible Diffie-Hellman exchange.

Each of the protocols is now defined.

2.1. PAX_STD Protocol

PAX_STD is a simple nonce-based authentication using the strong long-term key. The client and server each exchange 256 bits of random data, which is used to seed the PAX-KDF for generation of session keys. The randomly exchanged data in the protocol differs depending on whether a key update is being performed. If no key update is being performed, then let:

- A = X
- B = Y
- E = X || Y

Thus, A and B are 256-bit values and E is their 512-bit concatenation. To provide forward secrecy and security, let the following be true when a key update is being performed:

- A = g^X
- B = g^Y
- E = g^(XY)

Here A and B are Diffie-Hellman exponents whose length is determined by the Diffie-Hellman group size. The value A is data transmitted

from the server to the client, and B is data transmitted from the client to the server. The value E is the entropy computed by each that is used in Section 2.4 to perform key derivation.

The full protocol is as follows:

- PAX_STD-1 : client <- server : A
- PAX_STD-2 : client -> server : B, CID, MAC_CK(A, B, CID), [optional ADE]
- PAX_STD-3 : client <- server : MAC_CK(B, CID), [optional ADE]
- PAX-ACK : client -> server : [optional ADE]

See Section 2.3 for more information on the ADE component, and Section 2.4 for the key derivation process, including derivation of CK.

2.2. PAX_SEC Protocol

PAX_SEC is the high-security protocol designed to provide identity protection and support for provisioning. PAX_SEC requires a server-side public key, and public-key operations for every authentication.

PAX_SEC can be performed with and without key update. Let A, B, and E be defined as in the previous section.

The exchanges for PAX_SEC are as follows:

- PAX_SEC-1 : client <- server : M, PK or CertPK
- PAX_SEC-2 : client -> server : Enc_PK(M, N, CID)
- PAX_SEC-3 : client <- server : A, MAC_N(A, CID)
- PAX_SEC-4 : client -> server : B, MAC_CK(A, B, CID), [optional ADE]
- PAX_SEC-5 : client <- server : MAC_CK(B, CID), [optional ADE]
- PAX-ACK : client -> server : [optional ADE]

See Section 2.3 for more information on the ADE component, and Section 2.4 for the key derivation process, including derivation of CK.

Use of CertPK is optional in PAX_SEC; however, careful consideration should be given before omitting the CertPK. The following table describes the risks involved when using PAX_SEC without a certificate.

Certificate Mode	Provisioning	Identity Protection
No Certificate	MiTM offline dictionary attack	ID reveal attack
Self-Signed Certificate	MiTM offline dictionary attack	ID reveal attack
Certificate/PK Caching	MiTM offline dictionary attack	ID reveal attack during first auth
CA-Signed Certificate	secure mutual authentication	secure mutual authentication

Figure 2: Table of Different Security Modes

When using PAX_SEC to support provisioning with a weak key, use of a CA-signed certificate is RECOMMENDED. When not using a CA-signed certificate, the initial authentication is vulnerable to an offline man-in-the-middle (MiTM) dictionary attack.

When using PAX_SEC to support identity protection, use of either a CA-signed certificate or key caching is RECOMMENDED. Caching involves a client recording the public key of the EAP server and verifying its consistency between sessions, similar to Secure SHell (SSH) Protocol [RFC4252]. Otherwise, an attacker can spoof an EAP server during a session and gain knowledge of a client's identity.

Whenever certificates are used, clients MUST validate that the certificate's extended key usage, KeyPurposeID, is either "eapOverPPP" or "eapOverLAN" [RFC3280][RFC4334]. If the underlying EAP transport protocol is known, then the client MUST differentiate between these values. For example, an IEEE 802.11 supplicant SHOULD require KeyPurposeID == eapOverLAN. By not distinguishing, a client could accept as valid an unauthorized server certificate.

When using EAP-PAX with Wireless LAN, clients SHOULD validate that the certificate's wlanSSID extension matches the SSID of the network to which it is currently authenticating.

In order to facilitate discussion of packet validations, three client security policies for PAX_SEC are defined.

open
   Clients support both use of PK and CertPK. If CertPK is used, the client MUST validate the KeyPurposeID.

caching
: Clients save PK for each EAP server the first time it encounters the server, and SHOULD NOT authenticate to EAP servers whose public key has been changed. If CertPK is used, the client MUST validate the KeyPurposeID.

strict
: In strict mode, clients require servers to present a valid certificate signed by a trusted CA. As with the other modes, the KeyPurposeID MUST be validated.

Servers SHOULD support the PAX_SEC mode of operation, and SHOULD support both the use of PK and CertPK with PAX_SEC. Clients MUST support PAX_SEC, and MUST be capable of accepting both raw public keys and certificates from the server. Local security policy will define which forms of key or certificate authentications are permissible. Default configurations SHOULD require a minimum of the caching security policy, and MAY require strict.

The ability to perform key management on the AK is built in to EAP-PAX through the use of AK'. However, key management of the server public key is beyond the scope of this document. If self-signed certificates are used, the deployers should be aware that expired certificates may be difficult to replace when the caching security mode is used.

See Section 4 for further discussion on security considerations.

2.3. Authenticated Data Exchange

Messages PAX_STD-2, PAX_STD-3, PAX_SEC-4, PAX_SEC-5, and PAX_ACK contain optional component ADE. This component is used to convey authenticated data between the client and server during the authentication.

The Authenticated Data Exchanged (ADE) can be used in a variety of ways, including the implementation of channel bindings. Channel bindings allow link-layer network properties to be securely validated by the EAP client and server during the authentication session.

It is important to note that ADE is not encrypted, so any data included will not be confidential. However, since these packets are all protected by the Integrity Check Value (ICV), authenticity is guaranteed.

The ADE element consists of an arbitrary number of subelements, each with length and type specified. If the number and size of subelements is too large, packet fragmentation will be necessary. Vendor-specific options are supported. See Section 3.3.

Note that more than 1.5 round-trips may be necessary to execute a particular authenticated protocol within EAP-PAX. In this case, instead of sending an EAP-Success after receiving the PAX_ACK, the server can continue sending PAX_ACK messages with attached elements. The client responds to these PAX_ACK messages with PAX_ACK messages possibly containing more ADE elements. Such an execution could look something like the following:

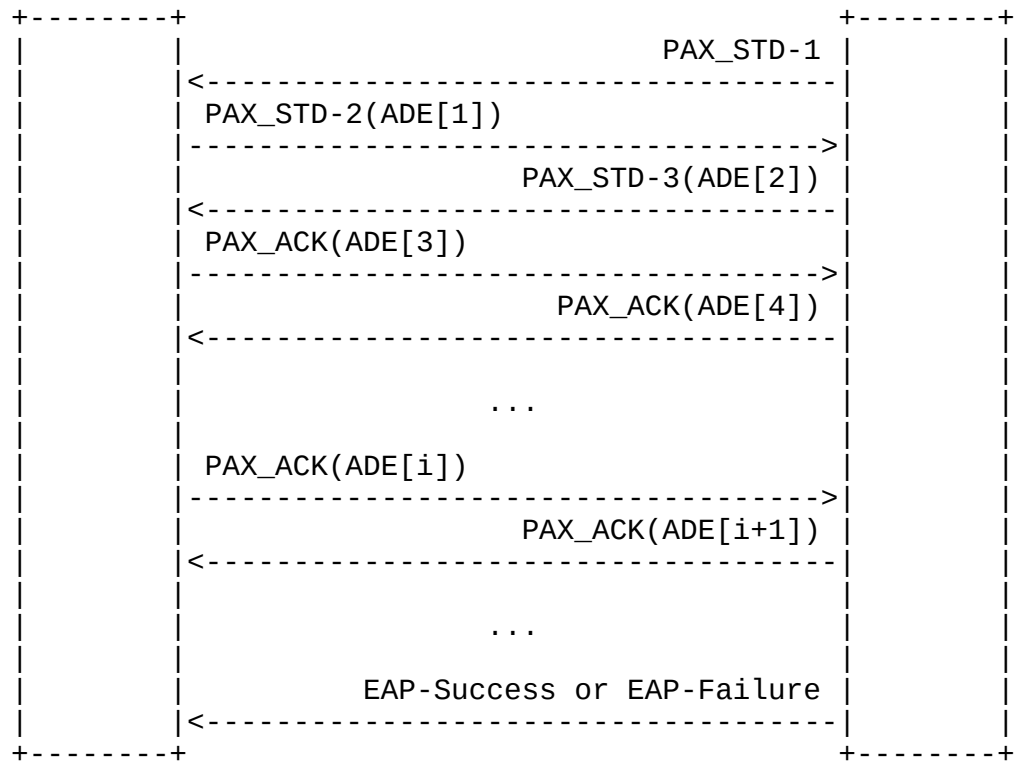

Figure 3: Extended Diagram of EAP-PAX Packet Exchanges

2.4. Key Derivation

Keys are derived independently of which authentication mechanism was used. The process uses the entropy value E computed as described above. Session and authentication keys are computed as follows:

- AK' = PAX-KDF-16(AK, "Authentication Key", E)
- MK = PAX-KDF-16(AK, "Master Key", E)

- CK = PAX-KDF-16(MK, "Confirmation Key", E)
- ICK = PAX-KDF-16(MK, "Integrity Check Key", E)
- MID = PAX-KDF-16(MK, "Method ID", E)
- MSK = PAX-KDF-64(MK, "Master Session Key", E)
- EMSK = PAX-KDF-64(MK, "Extended Master Session Key", E)
- IV = PAX-KDF-64(0x00^16, "Initialization Vector", E)

The IV is computed using a 16-octet NULL key. The value of AK' is only used to replace AK if a key update is being performed. The EAP Method ID is represented in ASCII as 32 hexadecimal characters without any octet delimiters such as colons or dashes.

The EAP Key Management Framework [IETF.KEY] recommends specification of key names and scope. The EAP-PAX Method-ID is the MID value computed as described above. The EAP peer name is the CID value exchanged in PAX_STD-2 and PAX_SEC-2. The EAP server name is an empty string.

2.5. Verification Requirements

In order for EAP-PAX to be secure, MACs must be properly verified each step of the way. Any packet with an ICV (see Section 3.4) that fails validation must be silently discarded. After ICV validation, the following checks must be performed:

PAX_STD-2
: The server MUST validate the included MAC, as it serves to authenticate the client to the server. If this validation fails, the server MUST send an EAP-Failure message.

PAX_STD-3
: The client MUST validate the included MAC, as it serves to authenticate the server to the client. If this validation fails, the client MUST send an EAP-Failure message.

PAX_SEC-1
: The client MUST validate PK or CertPK in a manner specified by its local security policy (see Section 2.2). If this validation fails, the client MUST send an EAP-Failure message.

PAX_SEC-2
: The server MUST verify that the decrypted value of M matches the value transmitted in PAX_SEC-1. If this validation fails, the server MUST send an EAP-Failure message.

PAX_SEC-3
The client MUST validate the included MAC, as it serves to prevent replay attacks. If this validation fails, the client MUST send an EAP-Failure message.

PAX_SEC-4
The server MUST validate the included MAC, as it serves to authenticate the client to the server. If this validation fails, the server MUST send an EAP-Failure message.

PAX_SEC-5
The client MUST validate the included MAC, as it serves to authenticate the server to the client. If this validation fails, the client MUST send an EAP-Failure message.

PAX-ACK
If PAX-ACK is received in response to a message fragment, the receiver continues the protocol execution. If PAX-ACK is received in response to PAX_STD-3 or PAX_SEC-5, then the server MUST send an EAP-Success message. This indicates a successful execution of PAX.

2.6. PAX Key Derivation Function

The PAX-KDF is a secure key derivation function used to generate various keys from the provided entropy and shared key.

PAX-KDF-W(X, Y, Z)

W length, in octets, of the desired output
X secret key used to protect the computation
Y public identifier for the key being derived
Z exchanged entropy used to seed the KDF

Let’s define some variables and functions:

- M_i = MAC_X(Y || Z || i), where i is an 8-bit unsigned integer
- L = ceiling(W/16)
- F(A, B) = first A octets of binary data B

We define PAX-KDF-W(X, Y, Z) = F(W, M_1 || M_2 || ... || M_L).

Consequently for the two values of W used in this document, we have:

- PAX-KDF-16(X, Y, Z) = MAC_X(Y || Z || 0x01)
- PAX-KDF-64(X, Y, Z) = MAC_X(Y || Z || 0x01) || MAC_X(Y || Z || 0x02) || MAC_X(Y || Z || 0x03) || MAC_X(Y || Z || 0x04)

The MAC used in the PRF is extensible and is the same MAC used in the rest of the protocol. It is specified in the EAP-PAX header.

3. Protocol Specification

In this section, the packet format and content for the EAP-PAX messages are defined.

EAP-PAX packets have the following structure:

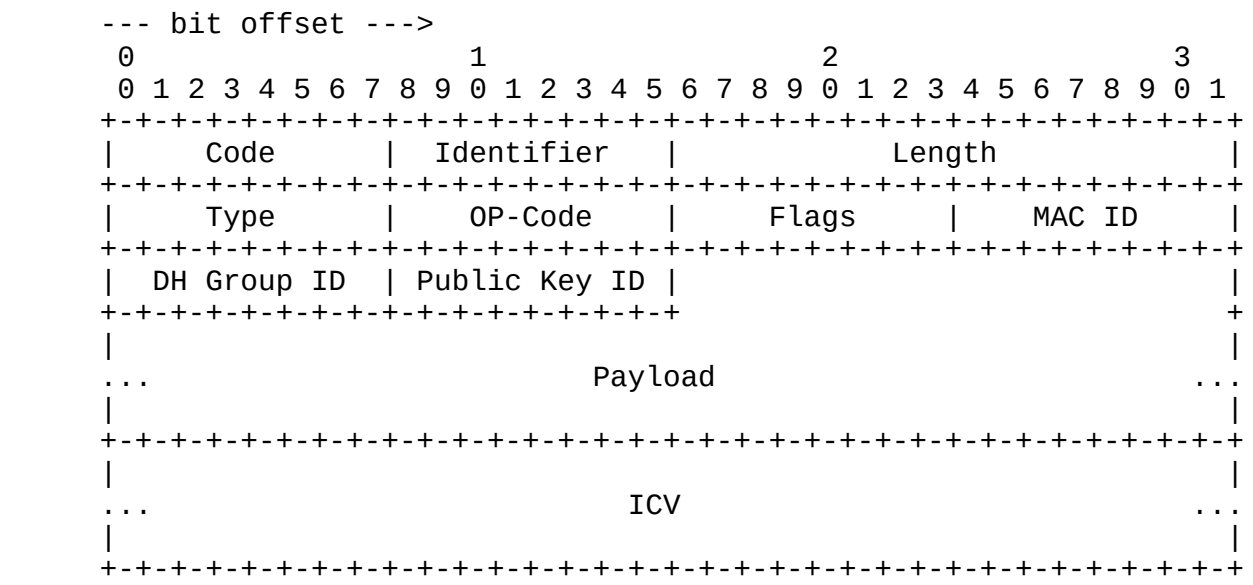

Figure 4: EAP-PAX Packet Structure

3.1. Header Specification

The Code, Identifier, Length, and Type fields are all part of the EAP header, and defined in [RFC3748]. IANA has allocated EAP Method Type 46 for EAP-PAX; thus, the Type field in the EAP header MUST be 46.

3.1.1. Op-Code

The OP-Code field is one of the following values:

- 0x01 : PAX_STD-1
- 0x02 : PAX_STD-2
- 0x03 : PAX_STD-3
- 0x11 : PAX_SEC-1
- 0x12 : PAX_SEC-2
- 0x13 : PAX_SEC-3
- 0x14 : PAX_SEC-4

- o 0x15 : PAX_SEC-5
- o 0x21 : PAX-ACK

3.1.2. Flags

The flags field is broken up into 8 bits each representing a binary flag. The field is defined as the Logical OR of the following values:

- o 0x01 : more fragments (MF)
- o 0x02 : certificate enabled (CE)
- o 0x04 : ADE Included (AI)
- o 0x08 - 0x80 : reserved

The MF flag is set if the current packet required fragmentation, and further fragments need to be transmitted. If a packet does not require fragmentation, the MF flag is not set.

When a payload requires fragmentation, each fragment is transmitted, and the receiving party responds with a PAX-ACK packet for each received fragment.

When using PAX_STD, the CE flag MUST be zero. When using PAX_SEC, the CE flag MUST be set if PAX_SEC-1 includes CertPK. It MUST NOT be set if PAX_SEC-1 includes PK. If CE is set in PAX_SEC-1, it MUST be set in PAX_SEC-2, PAX_SEC-3, PAX_SEC-4, and PAX_SEC-5. If either party detects an inconsistent value of the CE flag, he MUST send an EAP-Failure message and discontinue the session.

The AI flag indicates the presence of an ADE element. AI MUST only be set on packets PAX_STD-2, PAX_STD-3, PAX_SEC-4, PAX_SEC-5, and PAX_ACK if an ADE element is included. On packets of other types, ADE elements MUST be silently discarded as they cannot be authenticated.

3.1.3. MAC ID

The MAC field specifies the cryptographic hash used to generate the keyed hash value. The following are currently supported:

- o 0x01 : HMAC_SHA1_128 [FIPS198] [FIPS180]
- o 0x02 : HMAC_SHA256_128 [FIPS180]

3.1.4. DH Group ID

The Diffie-Hellman group field specifies the group used in the Diffie-Hellman computations. The following are currently supported:

- 0x00 : NONE (iff not performing a key update)
- 0x01 : 2048-bit MODP Group (IANA DH Group 14) [RFC3526]
- 0x02 : 3072-bit MODP Group (IANA DH Group 15) [RFC3526]
- 0x03 : NIST ECC Group P-256 [FIPS186]

If no key update is being performed, the DH Group ID field MUST be zero. Otherwise, the DH Group ID field MUST NOT be zero.

3.1.5. Public Key ID

The Public Key ID field specifies the cipher used to encrypt the client's EAP-Response in PAX_SEC-2.

The following are currently supported:

- 0x00 : NONE (if using PAX_STD)
- 0x01 : RSAES-OAEP [RFC3447]
- 0x02 : RSA-PKCS1-V1_5 [RFC3447]
- 0x03 : El-Gamal Over NIST ECC Group P-256 [FIPS186]

If PAX_STD is being executed, the Public Key ID field MUST be zero. If PAX_SEC is being executed, the Public Key ID field MUST NOT be zero.

When using RSAES-OAEP, the hash algorithm and mask generation algorithm used SHALL be the MAC specified by the MAC ID, keyed using an all-zero key. The label SHALL be null.

The RSA-based schemes specified here do not dictate the length of the public keys. DER encoding rules will specify the key size in the key or certificate [X.690]. Key sizes SHOULD be used that reflect the desired level of security.

3.1.6. Mandatory to Implement

The following ciphersuite is mandatory to implement and achieves roughly 112 bits of security:

- HMAC_SHA1_128
- IANA DH Group 14 (2048 bits)
- RSA-PKCS1-V1_5 (RECOMMEND 2048-bit public key)

The following ciphersuite is RECOMMENDED and achieves 128 bits of security:

- HMAC_SHA256_128
- IANA DH Group 15 (3072 bits)
- RSAES-OAEP (RECOMMEND 3072-bit public key)

3.2. Payload Formatting

This section describes how to format the payload field. Depending on the packet type, different values are transmitted. Sections 2.1 and 2.2 define the fields, and in what order they are to be concatenated. For simplicity and since many field lengths can vary with the ciphersuite, each value is prepended with a 2-octet length value encoded as an integer as described below. This length field MUST equal the length in octets of the subsequent value field.

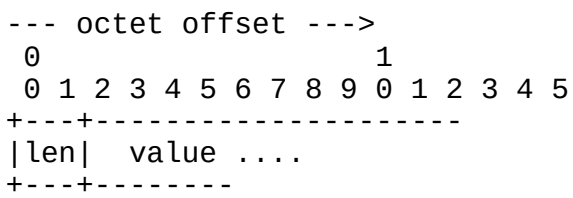

Figure 5: Length Encoding for Data Elements

All integer values are stored as octet arrays in network-byte order, with the most significant octet first. Integers are padded on the most significant end to reach octet boundaries.

Public keys and certificates SHALL be in X.509 format [RFC3280] encoded using the Distinguished Encoding Rules (DER) format [X.690].

Strings are not null-terminated and are encoded using UTF-8. Binary data, such as message authentication codes, are transmitted as-is.

MACs are computed by concatenating the specified values in the specified order. Note that for MACs, length fields are not included, though the resulting MAC will itself have a length field. Values are encoded as described above, except that no length field is specified.

To illustrate this process, an example is presented. What follows is the encoding of the payload for PAX_STD-2. The three basic steps will be computing the MAC, forming the payload, and encrypting the payload.

To create the MAC, we first need to form the buffer that will be MACed. For this example, assume that no key update is being done and HMAC_SHA1_128 is used such that the result will be a 16-octet value.

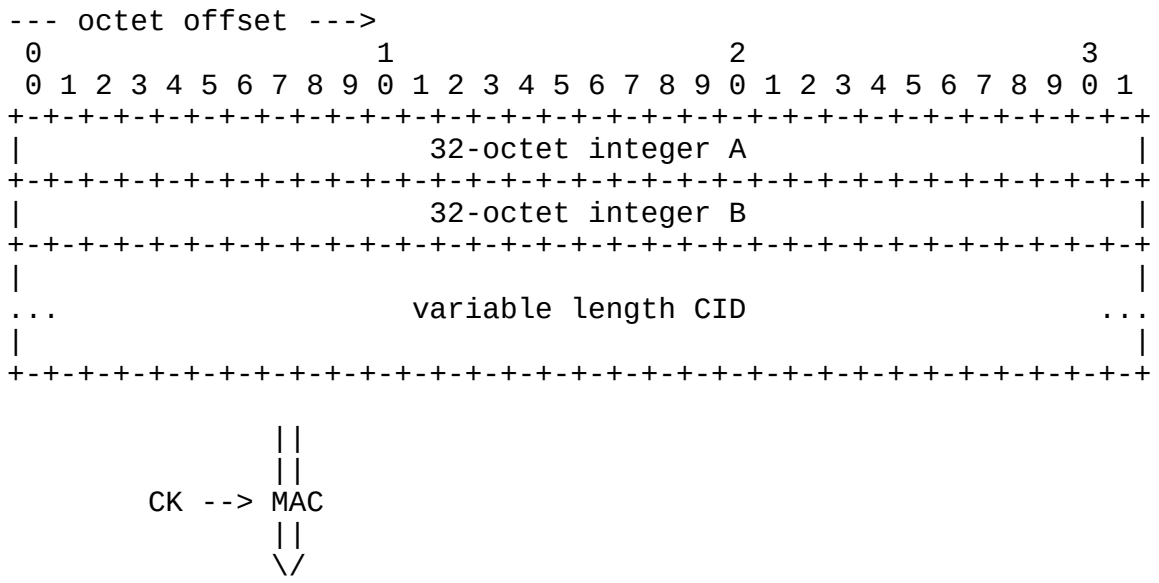

Figure 6: Example Encoding of PAX_STD-2 MAC Data

With this, we can now create the encoded payload:

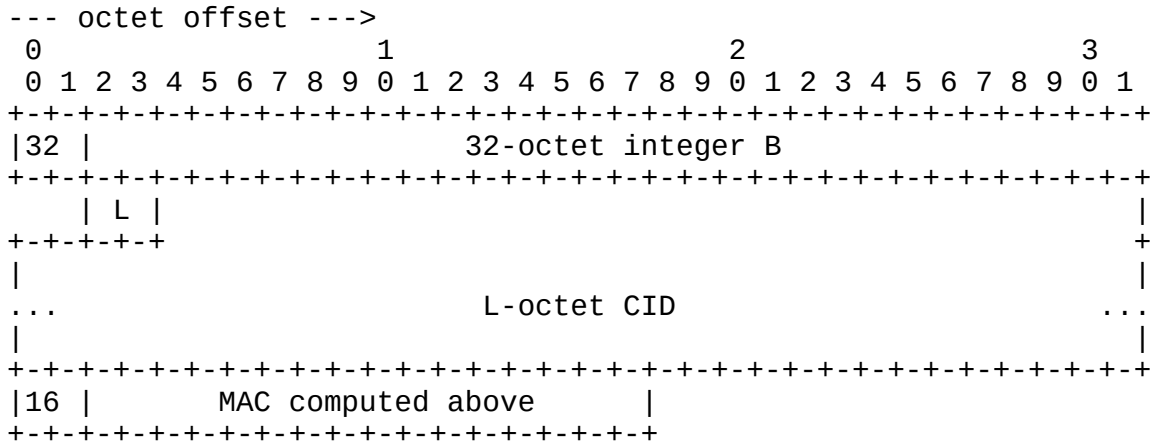

Figure 7: Example Encoding of PAX_STD-2 Packet

These 52+L octets are then attached to the packet as the payload. The ICV is then computed by MACing the packet headers and payload, and appended after the payload (see Section 3.4).

3.3. Authenticated Data Exchange (ADE)

This section describes the formatting of the ADE elements. ADE elements can only occur on packets of type PAX_STD-2, PAX_STD-3, PAX_SEC-4, PAX_SEC-5, and PAX_ACK. Values included in other packets MUST be silently ignored.

The ADE element is preceded by its 2-octet length L. Each subelement has first a 2-octet length Li followed by a 2-octet type Ti. The entire ADE element looks as follows:

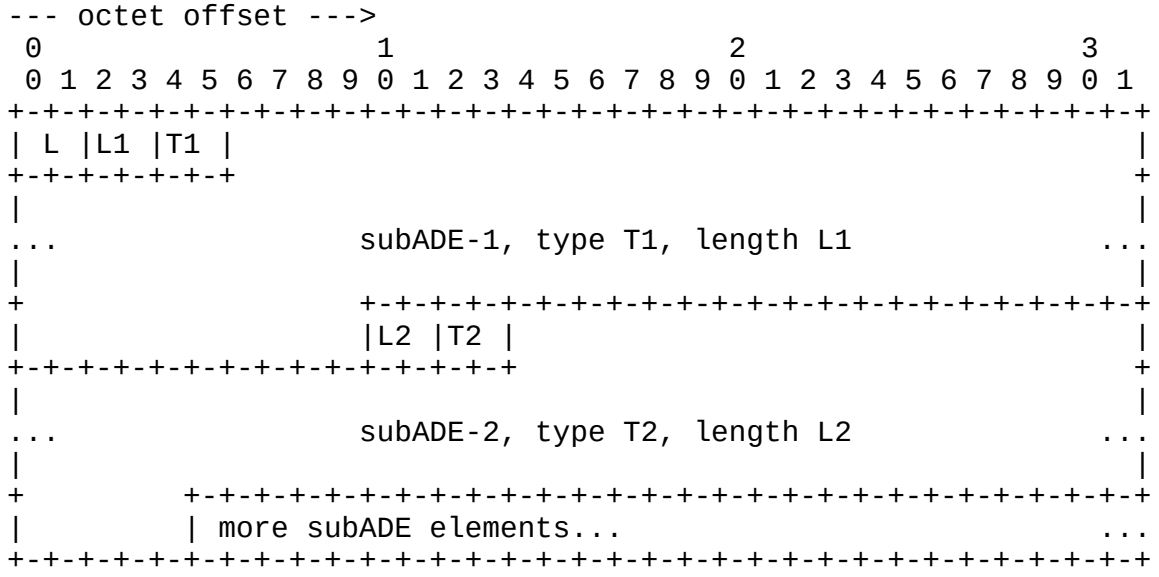

Figure 8: Encoding of ADE Components

The following type values have been allocated:

- 0x01 : Vendor Specific
- 0x02 : Client Channel Binding Data
- 0x03 : Server Channel Binding Data

The first three octets of a subADE utilizing type code 0x01 must be the vendor's Enterprise Number [RFC3232] as registered with IANA. The format for such a subADE is as follows:

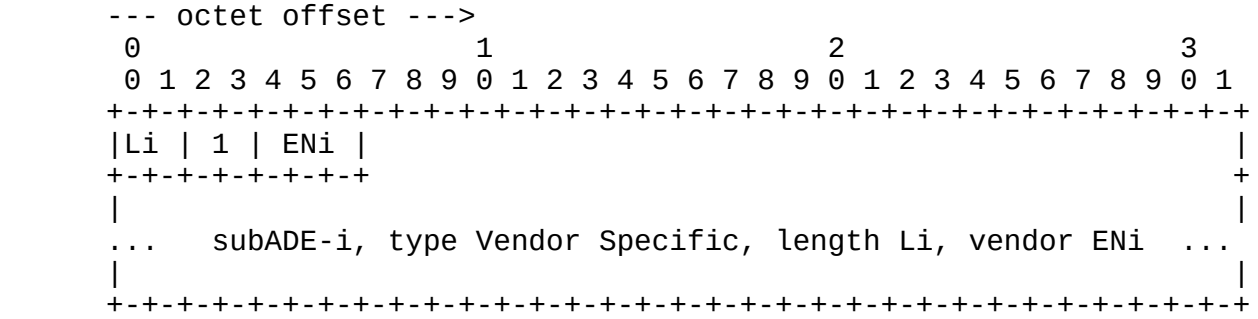

Figure 9: Encoding of Vendor-specific ADE

Channel binding subADEs have yet to be defined. Future IETF documents will specify the format for these subADE fields.

3.4. Integrity Check Value (ICV)

The ICV is computed as the MAC over the entire EAP packet, including the EAP header, the EAP-PAX header, and the EAP-PAX payload. The MAC is keyed using the 16-octet ICK, using the MAC type specified by the MAC ID in the EAP-PAX header. For packets of type PAX_STD-1, PAX_SEC-1, PAX_SEC-2, and PAX_SEC-3, where the MK has not yet been derived, the MAC is keyed using a zero-octet NULL key.

If the ICV field is incorrect, the receiver MUST silently discard the packet.

4. Security Considerations

Any authentication protocol, especially one geared for wireless environments, must assume that adversaries have many capabilities. In general, one must assume that all messages between the client and server are delivered via the adversary. This allows passive attackers to eavesdrop on all traffic, while active attackers can modify data in any way before delivery.

In this section, we discuss the security properties and requirements of EAP-PAX with respect to this threat model. Also note that the security of PAX can be proved using under the Random Oracle model.

4.1. Server Certificates

PAX_SEC can be used in several configurations. It can be used with or without a server-side certificate. Section 2.2 details the possible modes and the resulting security risk.

When using PAX_SEC for identity protection and not using a CA-signed certificate, an attacker can convince a client to reveal his username. To achieve this, an attacker can simply forge a PAX_SEC-1 message and send it to the client. The client would respond with a PAX_SEC-2 message containing his encrypted username. The attacker can then use his associated private key to decrypt the client's username. Use of key caching can reduce the risk of identity revelation by allowing clients to detect when the EAP server to which they are accustom has a different public key.

When provisioning with PAX_SEC and not using a CA-signed certificate, an attacker could first forge a PAX_SEC-1 message and send it to the client. The client would respond with a PAX_SEC-2 message. Using the decrypted value of N, an attacker could forge a PAX_SEC-3 message. Once the client responds with a PAX_SEC-4 message, an attacker can guess values of the weak AK and compute CK = PAX-KDF(AK, "Confirmation Key", g^XY). Given enough time, the attacker can obtain both the old AK and new AK' and forge a responding PAX_SEC-5.

4.2. Server Security

In order to maintain a reasonable security policy, the server should manage five pieces of information concerning each user, most obviously, the username and current key. In addition, the server must keep a bit that indicates whether the current key is weak. Weak keys must be updated prior to key derivation. Also, the server should track the date of last key update. To implement the coarse-grained forward secrecy, the authentication key must be updated on a regular basis, and this field can be used to expire keys. Last, the server should track the previous key, to prevent attacks where an adversary desynchronizes the key state by interfering with PAX-ACK packets. See Appendix B for more suggested implementation strategies that prevent key desynchronization attacks.

Since the client keys are stored in plaintext on the server, special care should be given to the overall security of the authentication server. An operating system-level attack yielding root access to an intruder would result in the compromise of all client credentials.

4.3. EAP Security Claims

This section describes EAP-PAX in terms of specific security terminology as required by [RFC3748].

4.3.1. Protected Ciphersuite Negotiation

In the initial packet from the server, the server specifies the ciphersuite in the packet header. The server is in total control of the ciphersuite; thus, a client not supporting the specified ciphersuite will not be able to authenticate. In addition, each client's local security policy should specify secure ciphersuites the client will accept. The ciphersuite specified in PAX_STD-1 and PAX_SEC-1 MUST remain the same in successive packets within the same authentication session. Since later packets are covered by an ICV keyed with the ICK, the server can verify that the originally transmitted ciphersuite was not altered by an adversary.

4.3.2. Mutual Authentication

Both PAX_STD and PAX_SEC authenticate the client and the server, and consequently achieve explicit mutual authentication.

4.3.3. Integrity Protection

The ICV described in Section 3.4 provides integrity protection once the integrity check key has been derived. The header values in the unprotected packets can be verified when an ICV is received later in the session.

4.3.4. Replay Protection

EAP-PAX is inherently designed to avoid replay attacks by cryptographically binding each packet to the previous one. Also the EAP sequence number is covered by the ICV to further strengthen resistance to replay attacks.

4.3.5. Confidentiality

With identity protection enabled, PAX_SEC provides full confidentiality.

4.3.6. Key Derivation

Session keys are derived using the PAX-KDF and fresh entropy supplied by both the client and the server. Since the key hierarchy is derived from the shared password, only someone with knowledge of that password or the capability of guessing it is capable of deriving the

session keys. One of the main benefits of PAX_SEC is that it allows you to bootstrap a strong shared secret using a weak password while preventing offline dictionary attacks.

4.3.7. Key Strength

Authentication keys are 128 bits. The key generation is protected by a Diffie-Hellman key exchange. It is believed that a 3000-bit MODP public-key scheme is roughly equivalent [RFC3766] to a 128-bit symmetric-key scheme. Consequently, EAP-PAX requires the use of a Diffie-Hellman group with modulus larger than 3000. Also, the exponent used as the private DH parameter must be at least twice as large as the key eventually generated. Consequently, EAP-PAX uses 256-bit DH exponents. Thus, the authentication keys contain the full 128 bits of security.

Future ciphersuites defined for EAP-PAX MUST contain a minimum of 128 bits of security.

4.3.8. Dictionary Attack Resistance

EAP-PAX is resistant to dictionary attacks, except for the case where a weak password is initially used and the server is not using a certificate for authentication. See Section 4.1 for more information on resistance to dictionary attacks.

4.3.9. Fast Reconnect

Although a specific fast reconnection option is not included, execution of PAX_STD requires very little computation time and is therefore bound primarily by the latency of the Authentication, Authorization, and Accounting (AAA) server.

4.3.10. Session Independence

This protocol easily achieves backward secrecy through, among other things, use of the PAX-KDF. Given a current session key, attackers can discover neither the entropy used to generate it nor the key used to encrypt that entropy as it was transmitted across the network.

This protocol has coarse-grained forward secrecy. Compromised session keys are only useful on data for that session, and one cannot derive AK from them. If an attacker can discover AK, that value can only be used to compromise session keys derived using that AK. Reasonably frequent password updates will help mitigate such attacks.

Session keys are independently generated using fresh nonces for each session, and therefore the sessions are independent.

4.3.11. Fragmentation

Fragmentation and reassembly is supported through the fragmentation flag in the header.

4.3.12. Channel Binding

EAP-PAX can be extended to support channel bindings through the use of its subADE fields.

4.3.13. Cryptographic Binding

EAP-PAX does not include any cryptographic binding. This is relevant only for tunneled methods.

4.3.14. Negotiation Attack Prevention

EAP is susceptible to an attack where an attacker uses NAKs to convince an EAP client and server to use a less secure method, and can be prevented using method-specific integrity protection on NAK messages. Since EAP-PAX does not have suitable keys derived for this integrity protection at the beginning of a PAX conversation, this is not included.

5. IANA Considerations

This document requires IANA to maintain the namespace for the following header fields: MAC ID, DH Group ID, Public Key ID, and ADE type. The initial namespace populations are as follows.

MAC ID Namespace:

- 0x01 : HMAC_SHA1_128
- 0x02 : HMAC_SHA256_128

DH Group ID Namespace:

- 0x00 : NONE
- 0x01 : IANA DH Group 14
- 0x02 : IANA DH Group 15
- 0x03 : NIST ECC Group P-256

Public Key ID Namespace:

- 0x00 : NONE
- 0x01 : RSAES-OAEP
- 0x02 : RSA-PKCS1-V1_5
- 0x03 : El-Gamal Over NIST ECC Group P-256

ADE Type Namespace:

- 0x01 : Vendor Specific
- 0x02 : Client Channel Binding Data
- 0x03 : Server Channel Binding Data

Allocation of values for these namespaces shall be reviewed by a Designated Expert appointed by the IESG. The Designated Expert will post a request to the EAP WG mailing list (or a successor designated by the Designated Expert) for comment and review, including an Internet-Draft. Before a period of 30 days has passed, the Designated Expert will either approve or deny the registration request and publish a notice of the decision to the EAP WG mailing list or its successor, as well as informing IANA. A denial notice must be justified by an explanation and, in the cases where it is possible, concrete suggestions on how the request can be modified so as to become acceptable.

6. Acknowledgments

The authors would like to thank Jonathan Katz for discussion with respect to provable security, Bernard Aboba for technical guidance, Jari Arkko for his expert review, and Florent Bersani for feedback and suggestions. Finally, the authors would like to thank the Defense Information Systems Agency for initially funding this work.

7. References

7.1. Normative References

[FIPS180] National Institute for Standards and Technology, "Secure Hash Standard", Federal Information Processing Standard 180-2, August 2002.

[FIPS186] National Institute for Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standard 186, May 1994.

[FIPS198] National Institute for Standards and Technology, "The Keyed-Hash Message Authentication Code (HMAC)", Federal Information Processing Standard 198, March 2002.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3232] Reynolds, J., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, January 2002.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

[RFC4334] Housley, R. and T. Moore, "Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN)", RFC 4334, February 2006.

[X.690] International Telecommunications Union, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", Data Networks and Open System Communication Recommendation X.690, July 2002.

7.2. Informative References

[IETF.KEY] Aboba, B., Simon, D., Arkko, J., Eronen, P., and H. Levkowetz, "Extensible Authentication Protocol (EAP) Key Management Framework", Work in Progress.

[IEEE.80211] Institute of Electrical and Electronics Engineers, "Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", IEEE Standard 802.11-1997, 1997.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

[RFC4017] Stanley, D., Walker, J., and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, March 2005.

[RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

Appendix A. Key Generation from Passwords

If a 128-bit key is not available to bootstrap the authentication process, then one must be generated from some sort of weak preshared key. Note that the security of the hashing process is unimportant, as long as it does not significantly decrease the password's entropy. Resistance to dictionary attacks is provided by PAX_SEC. Consequently, computing the SHA-1 of the password and truncating the output to 128 bits is RECOMMENDED as a means of converting a weak password to a key for provisioning.

When using other preshared credentials, such as a Kerberos Data Encryption Standard (DES) key, or an MD4-hashed Microsoft Challenge Handshake Authentication Protocol (MSCHAP) password, to provision clients, these keys SHOULD still be put through SHA-1 before being used. This serves to protect the credentials from possible compromise, and also keeps things uniform. As an example, consider provisioning using an existing Kerberos credential. The initial key computation could be SHA1_128(string2key(password)). The KDC, storing string2key(password), would also be able to compute this initial key value.

Appendix B. Implementation Suggestions

In this section, two implementation strategies are discussed. The first describes how best to implement and deploy EAP-PAX in an enterprise network for IEEE 802.11i authentication. The second describes how to use EAP-PAX for device authentication in a 3G-style mobile phone network.

B.1. WiFi Enterprise Network

For the purposes of this section, a wireless enterprise network is defined to have the following characteristics:

- Users wish to obtain network access through IEEE 802.11 access points.
- Users can possibly have multiple devices (laptops, PDAs, etc.) they wish to authenticate.
- A preexisting authentication framework already exists, for example, a Microsoft Active Directory domain or a Kerberos realm.

Two of the biggest challenges in an enterprise WiFi network is key provisioning and support for multiple devices. Consequently, it is recommended that the client's Network Access Identifier (NAI) have

the format username/KID@realm, where KID is a key ID that can be used to distinguish between different devices.

The client's supplicant can use a variety of sources to automatically generate the KID. Two of the better choices would likely be the computer's NETBIOS name, or local Ethernet adapter's MAC address. The wireless adapter's address may be a suboptimal choice, as the user may only have one PCCARD adapter for multiple systems.

With an authentication system already in place, there is a natural choice for the provisioned key. Clients can authenticate using their preexisting password. When the server is presented with a new KID, it can create a new key record on the server and use the user's current password as the provisioned key. For example, for Active Directory, the supplicant could use Microsoft's NtPasswordHash function to generate a key verifiable by the server. It is suggested that this key then be fed through SHA1_128 before being used in a non-Microsoft authentication protocol.

After a key update, the server should keep track of both the old and new authentication keys. When two keys exist, the server should attempt to use both to validate the MACs on transmitted packets. Once a client successfully authenticates using the new key, the server should discard the old key. This prevents desynchronization attacks.

B.2. Mobile Phone Network

In a mobile phone system, we no longer need to worry about supporting multiple keys per identity. Presumably, each mobile device has a unique identity. However, if multiple devices per identity are desired, a method similar to that presented in Section B.1 could be used.

Provisioning could easily be accomplished by issuing customers a 6-digit PIN they could type into their phone's keypad.

Authors' Addresses

T. Charles Clancy
DoD Laboratory for Telecommunications Sciences
8080 Greenmeade Drive
College Park, MD 20740
USA

EMail: clancy@ltsnet.net

William A. Arbaugh
University of Maryland
Department of Computer Science
College Park, MD 20742
USA

EMail: waa@cs.umd.edu

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.