PI722a
VI Headers

Jeff Koftinoff <jeff.koftinoff@ieee.org>
Oct 8 2012 (a)
Considerations

• We want to be able to have optional cryptographic signing (via ECC) as well as encryption (via ECC and AES128) in P1722a

• IEEE Std 1722-2011 common header fields are not extendable and have no reserved bits that we can use

• Adding a ‘version 1’ header allows us to make a new AVTPDU common header (1722-2011 Figure 5.1)
Multiple Payloads

• We need multiple payloads in the Ethernet Frame:
  • The signature meta data (key guid, crypto type)
  • The encryption meta data (key guid, encryption type)
  • The subtype header and payload, either encrypted or not
  • Future expansion
  • The signature

• A payload may be both encrypted with one type of crypto, such as AES128 and signed with another type of crypto, such as public key ECC at the same time.
Header and Payload Ordering

• Payload/Header order matters so that we can enable hardware engines to process the packet ‘in flight’

• Encryption and Signature crypto type and key_guid fields need to come first to allow crypto engine initialization

• The Signature payload itself needs to come last to allow for the validation ‘in flight’
Header and Payload Alignment

• Even though a network frame may contain an optional tag, it is still advantageous to have header fields aligned to quadlets from the start of the payload in order to allow high bandwidth parallelized packet inspection in VHDL.

• I suggest that extension header offsets would be specified in counts of quadlets instead of octets, since if they were specified in octets and still require alignment then this is additional validation that needs to be done.

• We want to reduce the amount of parsing required so that hardware can find the appropriate offset of the data that it needs.

• We don’t want to have length fields in the beginning of the packet since the signature or encrypted payloads may be variable length and need to be able to be calculated ‘in flight’.

• However we don’t want to introduce a big adverse impact on the bandwidth available for payloads so we want the v1 common header to be minimal size.
Version 1 AVTPDU common header

- extension 0 is for signature crypto_type and key_guid
- extension 1 is for encryption crypto_type and key_guid
- extension 2 is for payload (either unencrypted subtype data or encrypted bitstream)
- extension 3 is for a signature blob
ext_layout is always 0 which means that there are four header extensions available. This could be considered a ‘sub-version field’ which could be extended in the future.

e0, e1, e2 and e3 are the extension enable bit. For instance, if the e0 bit is set to one, then this means e0_type, e0_flags, and extension0_quadlet_offset are valid.

e0_flags, e1_flags, e2_flags, and e3_flags are bit flags specific to the associated extension type.

The extension quadlet offset fields specify the offset in quadlets from the beginning of the AVTPDU common header.
Extension Type Codes

- 0: Unencrypted IEEE 1722 subtype data
- 1: Encrypted IEEE 1722 subtype data
- 2: Signature Blob
- 3: AES128 crypto key_guid
- 4: AES256 crypto key_guid
- 5: ECC crypto public key_guid
- 6: RSA crypto public key_guid

For all these extension type codes, the associated flags field is set to 0.

Since we want readers to be efficient at knowing where in the frame to look for the data, we have the following restrictions:

Extension 0, the signature extension, shall be type 5 (ECC key) or 6 (RSA key).

Extension 1, the encryption extension, shall be type 5 (ECC key) or 6 (RSA key).

Extension 2, the payload extension, shall be type 0 (unencrypted 1722 subtype data) or type 1 (encrypted 1722 subtype data).

Extension 3, the Signature blob extension shall be type 2 (signature blob).
Extension type 0: Plaintext 1722 Subtype Payload

The subtype’s type_specific_data is in the same relative location as it is in IEEE 1722-2011 Version 0 common header format.

Note: The ‘CD’, ‘subtype’ and ‘SV’ fields are duplicated here.
Extension type 1: Encrypted 1722 Subtype Payload

This block contains the Extension type 0 data, encrypted with the algorithm selected by extension 1 and includes all setup codes required for the crypto. When successfully decrypted, the result shall be identical to the Extension type 0 format.
Extension type 2: Signature Blob

This block contains the actual signature calculated with the key_guid from extension 0 and the entire frame’s data including SA, DA, Ethertype, Tag, V1 header and all extension headers up to this point in the PDU.
Extension types 2,3,4,5 (key_guid)

with flags set to zero, the key_guid specifies the eui64 ‘name’ of the key used
Example PDU -

- P1722.1 D21 ACMPDU in a P1722a Version 1 header, signed with ECC key and not encrypted
Example PDU -

- P1722.1 D21 ACMPDU in a P1722a Version 1 header, signed with ECC key and encrypted with AES128

```
0  1  2  3
  4  5  6  7
  8  9 10
0  1  2  3  4  5  6  7  8  9 10 11

CD (1) subtype (7C) SV version (1) ext_layout (0) e0 (1) e1 (1) e2 (1) e3 (1) reserved

04  e0_type (5)  e0_flags (0)  extension0_quadlet_offset (3) e1_type (3)  e1_flags (0)  extension1_quadlet_offset (5)
08  e2_type (1)  e2_flags (0)  extension2_quadlet_offset (7) e3_type (2)  e3_flags (0)  extension3_quadlet_offset (ZZ/4)
12  Key guid  sig_key_guid
16  Key guid  enc_key_guid
20  Key guid  enc_key_guid
24  Key guid  enc_key_guid
28  encrypted bitstream of Unencrypted extension 2, including all appropriate crypto setup codes
ZZ  Extension 3
28  Extension 3

Signature data appropriate for selected signature algorithm/key
```