Diameter End-to-End Security: Keyed Message Digests, Digital Signatures, and Encryption

draft-korhonen-dime-e2e-security-01
Jouni Korhonen, Hannes Tschofenig
Dime WG, IETF#85
Overview

- Background
- Requirements
- Strawman solutions proposal
- Two aspects:
  - Authentication and Key Exchange
  - Actual AVP protection
- Changes from -00 to -01
Background

- Diameter has no end-to-end security framework at the moment. Acknowledged in RFC 6733.

- Folks deploying (=telco camp e.g., 3GPP and GSMA) large Diameter networks for roaming purposes realized that their security assumptions are not met. Solutions are needed now!

- Bilateral site-to-site VPNs with all your roaming partners does not scale in a long run and one loses the possible benefits of 3rd party “roaming proxies”.
Requirements

- Provide end-to-end security properties to Diameter on top of existing hop-by-hop security model.
  - End-to-end is between two nodes with any number of intermediates in between. This allows “site-to-site” type of deployments as well.

- Works with existing request routing and through proxy agents.

- Decouple key management from end-to-end AVP protection.

- Offer both integrity and confidentiality protection.

- Easy to integrate into existing Diameter applications (integrity protection).
Requirements – two deployment cases

Site-to-site

End-to-end

Realm example.com

Roaming network

Realm example.net

Protected AVPs

End-to-end protection over this path

Realm example.com

Roaming network

Realm example.net

Protected AVPs

End-to-end protection over this path

Diameter Node
E2E security
Aware

Edge Agent
E2E security
Aware

Diameter Node
NOT security
Aware

Edge Agent
E2E security
Aware

Diameter Node
NOT security
Aware
Strawman Proposal in
draft-korhonen-dime-e2e-security-01

- This solution focuses on protecting Diameter AVPs. To offer the functionality two AVPs are defined:
  - Signed-Data (octet string) for integrity protection of one or more AVPs.
  - Encrypted-Data (octet string) for confidentiality protection of one or more AVPs.

- We selected JSON-based approach:
  - JSON Web signature (JWS) for integrity protection.
  - JSON Web Encryption (JWE) for confidentiality protection.
  - Encoding is “Diameter friendly” – not JSON style text strings.
  - Reuses JSON IANA registries.

- Not tied to a specific Diameter application.

- Authentication and key management is not part of this proposal:
  - Likely that “one size fits all” approach will not work due to different deployment environments
Signed-Data AVP

- The AVP carries JSON Web Signature (JWS) of one or more of AVPs. Each protected AVP is hashed and the hash is included into the JWS payload.

- Hashed AVPs are linked to “originals” using their AVP Code. If there are multiple instances of the same AVP, you hash them all and do one by one verification -> allows for rearranging AVPs and detection of addition/removal/modification of AVPs.

- Both JWS Payload and signature use the same hash algorithm of the cryptographic algorithm indicated in the JWS Header.

- Can be included into existing Diameter applications.
**Encrypted-Data AVP**

- The AVP carries JSON Web Encryption (JWE) data structure and the JWE Payload embeds of one or more protected AVPs.

- Cannot be used with existing Diameter applications since encrypted AVPs are embedded inside the Encrypted-Data AVP(s).
Error Handling

- **Transient failures**
  - `DIAMETER_KEY_UNKNOWN` – A Signed-Data or an Encrypted-Data AVP is received that was generated using a key that cannot be found in the key store. To recover a new end-to-end key establishment procedure may need to be invoked.
  
  - `DIAMETER_HEADER_NAME_ERROR` (TBD12 – This error code is returned when a Header Parameter Name is not understood in the JWS-Header AVP or in the JWE-Header AVP.

- **Permanent failures**
  - `DIAMETER_DECRIPTION_ERROR` – This error code is returned when an Encrypted-Data AVP is received and the decryption fails for an unknown reason.
  
  - `DIAMETER_SIGNATURE_ERROR` – This error code is returned when a Signed-Data AVP is received and the verification fails for an unknown reason.
Changes from -00 to -01

- Clarification that both end-to-end and site-to-site approaches are in scope.

- Reworked the encoding of protected AVPs. They are now more Diameter like and compact. Still using JSON framework.

- New DIAMETER_HEADER_NAME_ERROR error code added.
Example of signature..

Signed-Data ::= < AVP Header: TBD1 >
 { JWS-Header }  
* { JWS-AVP-Payload }  
 { JWS-Signature }  
* [ AVP ]

The JWS Header used in this example is:

{"typ":"JWT",
 "alg":"HS256",
 "kid":"abc123"
}
Signed-Data Grouped AVP:
0x000000nnn  // Signed-Data code 'nnn'
0x000000e8   // Flags=0, Length=232(8+49+3+44+44+44+40)

JWS Header encoded into the JWS-Header AVP:
0x000000xxx  // JWS-Header code 'xxx'
0x00000031   // Flags=0, Length=49
'{"typ":"JWT","alg":"HS256","kid":"abc123"}'  // 41
0x00,0x00,0x00  // 3 octets padding

JWS Payload encoded into three JWS-AVP-Payload AVPs:
0x00000zzz  // JWS-AVP-Payload code 'zzz'  <--+
0x0000002c  // Flags=0, Length=44            |
0x00000107  // 263, Session-Id, 4 octets     |
0xca8362ed,0x69a32ffb // 256 bits hash of    |
0x9092ca98,0x745239da // Session-id         |
0x6960af73,0x6386bc38 |
0x407e518b,0xe4760548 |
0x00000zzz  // JWS-AVP-Payload code 'zzz'  |
0x0000002c  // Flags=0, Length=44            |
0x00000108  // 264, Origin-Host, 4 octets     |
0x64b52a15,0xa75a8157 // 256 bits hash of    |
0x151993a6,0xb9839866 // Origin-Realm       |
0x3b94afa3,0x85568552 |
0x46602ccc,0x3f9d9a77 |
0x00000zzz  // JWS-AVP-Payload code 'zzz'  |
0x0000002c  // Flags=0, Length=44            |
0x00000128  // 296, Origin-Realm, 4 octets    |
0x3c7c0b17,0x4a1c58d0 // 256 bits hash of    |
0xdc2844a3,0x28580385 // Origin-Realm       |
0x25eb08b0,0xeb20c941 |
0xcd52f74c,0xf55ae9ab  //                <--+

JWS Signature encoded into the JWS-Signature AVP:
0x000000yyy  // JWS-Signature code 'yyy'
0x00000028   // Flags=0, Length=40
0x70ec221e,0xe03000ec1,0xb7ce968d,0x6ec6ad9e
0x8afbe983,0x2b0e331c,0xe1f51ac,0xf9af0188

Individual AVP hash
Signature over this binary blob
Questions? Comments?

- First: is the end-to-end AVP protection **framework** approach feasible (forget JSON at this point)?

- Second: is **reusing** JSON ideas a feasible approach (forget encoding details at this point)?

- Third: does the WG think this I-D is a good **starting point**?