Encrypted Payloads in SUIT Manifests

Hannes Tschofenig , Russ Housley , Brendan Moran ,
David Brown , Ken Takayama

Status

- Draft updated in time for the deadline: https://datatracker.ietf.org/doc/html/draft-ietf-suit-firmware-encryption-19
- Diff shows a lot of changes.
- Reasons:
  - All examples have been updated and extended (which required a fair amount of implementation effort).
  - Feedback received during the WGLC has been incorporated.
  - Document has been re-structured for better readability.
- It is time to advance the document to the IESG.

Example Updates

- All examples are generated with python-cwt and

```
from cwt import COSE, COSEMessage, COSEKey, Recipient

plaintext = b"This is a real firmware image."
a128gcm_key = COSEKey.from_symmetric_key(
   bytes.fromhex("15F785B5C931414411B4B71373A9C0F7"), alg="A128GCM")
a128kw_key = COSEKey.from_symmetric_key("a" * 16, alg="A128KW", kid="kid-1")
r = Recipient.new(unprotected={"alg": "A128KW", "kid": a128kw_key.kid}, sender_key=a128kw_key)
sender = COSE.new()
encoded = sender.encode_and_encrypt(
   plaintext,
   a128gcm_key,
   protected={"alg": "A128GCM"},
   unprotected={"iv": bytes.fromhex("F14AAB9D81D51F7AD943FE87AF4F70CD")},
   recipients=[r])

recipient = COSE.new()
decrypted_payload = recipient.decode(encoded, keys=[a128kw_key])
assert plaintext == decrypted_payload
```

(e.g.) Encrypting and decrypting the payload with AES-KW + AES-GCM

- validated with ruby-cddl tool

```
$ cddl draft-ietf-suit-firmware-encryption.cddl validate suit-encryption-info-aes-kw-aes-gcm.cose
```

(e.g.) Checking the CDDL grammar and the binary match

You can see them in https://github.com/suit-wg/suit-firmware-encryption/tree/main/examples

Reflecting Review from Christian Amsüss

- It was not clear that the reason why the structure MUST be a zero-length byte string.

- For use with ciphers that do not provide integrity protection, such as AES-CTR and AES-CBC, - the Enc_structure MUST NOT be used and the protected header in the - SUIT_Encryption_Info_ESDH structure MUST be a zero-length byte string.
+ Some ciphers provide confidentiality without integrity protection, such as AES-CTR and AES-CBC. + For these ciphers the Enc_structure, shown in Figure 8, MUST NOT be used because the Additional + Authenticated Data (AAD) byte string is only consumable by AEAD ciphers. Hence, the AAD + structure is not supplied to the API of those ciphers and the protected header in the + SUIT_Encryption_Info structure MUST be a zero-length byte string.

- "More and more flashes are protected and are written at page level"

- re-writes data at byte level (often 4-bytes).
+ re-writes data at the byte level (often 4-bytes) or larger units.

https://github.com/suit-wg/suit-firmware-encryption/commit/070f7184cd452bc4612ebfbe6bdde85ca77ace17

Reflecting Review from Ruud Derwig

- Added a remark that beside confidentiality of the binary, confidentiality of the sources may be required.

```
+ Beside confidentiality of the binary, confidentiality of the sources (e.g. in case of open source
+ software) may be required as well to prevent reverse engineering and/or reproduction of
+ the binary firmware.
```

- "Is the goal protection of images during transportation, or when at rest stored inside a device?"

```
+ The goal of this specification is to protect payloads during transportation end-to-end, and at rest
+ when stored inside a device. Since many of the devices today do not offer hardware-based,
+ on-the-fly decryption of code stored in flash memory, it may be necessary to decrypt and
+ store firmware images in on-chip flash before code can be executed. Since devices with
+ hardware-based, on-the-fly decryption become more common, the goal of accomplishing
+ confidentiality at rest may be better accomplished.
```

https://github.com/suit-wg/suit-firmware-encryption/commit/018cad5a7cb2759bf102cac9d84052d9d9162f4a

Reflecting Review from Michael Rechardson

- Added a mention on Multiple KEKs

+ To further reduce the attack surface it may be beneficial use
+ different long-term keys for the encryption of different types
+ of payloads. For example, KEK_1 may be used with
+ an AES-KW content key distribution method to encrypt
+ a firmware image while KEK_2 would be used
+ to encrypt configuration data.

- Added a guidance for use case and key configuration in table-like format →

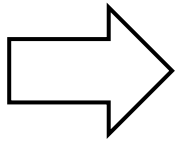
A large part of this document is focused on the content key distribution and two methods are utilized, namely AES Key Wrap (AES-KW) and Ephemeral-Static Diffie-Hellman (ES-DH). In this table we summarize the main properties with respect to their deployment:

Number of Long-Term Keys	Same key for all devices	One key per device	One Key per device
Number of Content Encryption Keys (CEKs)	Single CEK per payload shared with all devies	Single CEK per payload shared with all devies	One CEK per payload encryption transaction per device
Use Case	Legacy Usage	Efficient Payload Distribution	Point-to-Point Payload Distribution
Recommended?	No, bad practice	Yes	Yes

https://github.com/suit-wg/suit-firmware-encryption/commit/fc4de69353132223ecf27ccf67e7eb07380ba237
https://github.com/suit-wg/suit-firmware-encryption/commit/018cad5a7cb2759bf102cac9d84052d9d9162f4a

Re-structuring for better readability

- Before: Examples have been scattered into different levels
- After: Examples locate in Content Encryption section for each Key Distribution method



Ready for submitting to IESG?

- Special thanks to
  - Daisuke Ajitomi and Carsten Bormann for their powerful tools
  - Christian Amsüss, Ruud Derwig and Michael Rechardson for their insightful reviews