Workgroup: SUIT
Internet-Draft:
draft-ietf-suit-firmware-encryption-03
Published: 7 March 2022
Intended Status: Standards Track
Expires: 8 September 2022

Authors: H. Tschofenig   R. Housley   B. Moran
Arm Limited   Vigil Security   Arm Limited

Firmware Encryption with SUIT Manifests

Abstract

This document specifies a firmware update mechanism where the firmware image is encrypted. Firmware encryption uses the IETF SUIT manifest with key establishment provided by the hybrid public-key encryption (HPKE) scheme and the AES Key Wrap (AES-KW) with a pre-shared key-encryption key. Encryption of the firmware image is encrypted using AES-GCM or AES-CCM.

Status of This Memo

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Table of Contents

1. Introduction

Vulnerabilities with Internet of Things (IoT) devices have raised the need for a reliable and secure firmware update mechanism that is also suitable for constrained devices. To protect firmware images the SUIT manifest format was developed [I-D.ietf-suit-manifest]. The SUIT manifest provides a bundle of metadata about the firmware for an IoT device, where to find the firmware image, and the devices to which it applies.

The SUIT information model [RFC9124] details the information that has to be offered by the SUIT manifest format. In addition to offering protection against modification, which is provided by a digital signature or a message authentication code, the firmware image may also be afforded confidentiality using encryption.

Encryption prevents third parties, including attackers, from gaining access to the firmware binary. Hackers typically need intimate knowledge of the target firmware to mount their attacks. For example, return-oriented programming (ROP) requires access to the binary and encryption makes it much more difficult to write exploits.

The SUIT manifest provides the data needed for authorized recipients of the firmware image to decrypt it. The firmware image is encrypted using a symmetric key. This symmetric cryptographic key is established for encryption and decryption, and that key can be applied to a SUIT manifest, firmware images, or personalization data, depending on the encryption choices of the firmware author.

A symmetric key can be established using a variety of mechanisms; this document defines two approaches for use with the IETF SUIT manifest, namely:

*hybrid public-key encryption (HPKE), and

*AES Key Wrap (AES-KW) using a pre-shared key-encryption key (KEK).

These choices reduce the number of possible key establishment options and thereby help increase interoperability between different SUIT manifest parser implementations.

The document also contains a number of examples.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document assumes familiarity with the IETF SUIT manifest [I-D.ietf-suit-manifest], the SUIT information model [RFC9124] and the SUIT architecture [RFC9019].

The terms sender and recipient are defined in [I-D.irtf-cfrg-hpke] and have the following meaning:

*Sender: Role of entity which sends an encrypted message.

*Recipient: Role of entity which receives an encrypted message.

Additionally, the following abbreviations are used in this document:

- Key Wrap (KW), defined in RFC 3394 [RFC3394] for use with AES.

- Key-encryption key (KEK), a term defined in RFC 4949 [RFC4949].

- Content-encryption key (CEK), a term defined in RFC 2630 [RFC2630].

- Hybrid Public Key Encryption (HPKE), defined in [I-D.irtf-cfrg-hpke].

3. Architecture

[RFC9019] describes the architecture for distributing firmware images and manifests from the author to the firmware consumer. It does, however, not detail the use of encrypted firmware images.

This document enhances the SUIT architecutre to include firmware encryption. Figure 1 shows the distribution system, which represents the firmware server and the device management infrastructure. The distribution system is aware of the individual devices to which a firmware update has to be delivered.

```
                                           +----------+
                                           |          |
                                           |  Author  |
                                           |          |
 +----------+                              +----------+
 |  Device  |---+                               |
 |(Firmware |   |                               | Firmware +
 | Consumer)|   |                               | Manifest
 +----------+   |                               |
                |                               |
                |                        +--------------+
                |                        |              |
 +----------+   |  Firmware + Manifest   | Distribution |
 |  Device  |---+------------------------|    System    |
 |(Firmware |   |                        |              |
 | Consumer)|   |                        |              |
 +----------+   |                        +--------------+
                |
                |
 +----------+   |
 |  Device  +---+
 |(Firmware |
 | Consumer)|
 +----------+
```

Figure 1: Firmware Encryption Architecture.

Firmware encryption requires the sender to know the firmware consumers and the respective credentials used by the key distribution mechanism. For AES-KW the KEK needs to be known and, in case of HPKE, the sender needs to be in possession of the public key of the recipient.

The firmware author may have knowledge about all devices that need to receive an encrypted firmware image but in most cases this will not be likely. The distribution system certainly has the knowledge about the recipients to perform firmware encryption.

To offer confidentiality protection for firmware images two deployment variants need to be supported:

*The firmware author acts as the sender and the recipient is the firmware consumer (or the firmware consumers).

*The firmware author encrypts the firmware image with the distribution system as the initial recipient. Then, the distribution system decrypts and re-encrypts the firmware image towards the firmware consumer(s). Delegating the task of re-encrypting the firmware image to the distribution system offers flexiblity when the number of devices that need to receive encrypted firmware images changes dynamically or when updates to KEKs or recipient public keys are necessary. As a downside, the author needs to trust the distribution system with performing the re-encryption of the firmware image.

Irrespectively of the two variants, the key distribution data (in form of the COSE_Encrypt structure) is included in the SUIT envelope rather than in the SUIT manifest since the manifest will be digitally signed (or MACed) by the firmware author.

Since the SUIT envelope is not protected cryptographically an adversary could modify the COSE_Encrypt structure. For example, if the attacker alters the key distribution data then a recipient will decrypt the firmware image with an incorrect key. This will lead to expending energy and flash cycles until the failure is detected. To mitigate this attack, the optional suit-cek-verification parameter is added to the manifest. Since the manifest is protected by a digital signature (or a MAC), an adversary cannot successfully modify this value. This parameter allows the recipient to verify whether the CEK has successfully been derived.

Details about the changes to the envelope and the manifest can be found in the next section.

4. SUIT Envelope and SUIT Manifest

This specification introduces two extensions to the SUIT envelope and the manifest structure, as motivated in Section 3.

The SUIT envelope is enhanced with a key exchange payload, which is carried inside the suit-protection-wrappers parameter, see Figure 2. One or multiple SUIT_Encryption_Info payload(s) are carried within the suit-protection-wrappers parameter. The content of the SUIT_Encryption_Info payload is explained in Section 5 (for AES-KW) and in Section 6 (for HPKE). When the encryption capability is used, the suit-protection-wrappers parameter MUST be included in the envelope.

```
SUIT_Envelope_Tagged = #6.107(SUIT_Envelope)
SUIT_Envelope = {
  suit-authentication-wrapper => bstr .cbor SUIT_Authentication,
  suit-manifest  => bstr .cbor SUIT_Manifest,
  SUIT_Severable_Manifest_Members,
  suit-protection-wrappers => bstr .cbor {
      *(int/str) => [+ SUIT_Encryption_Info]
  }
  * SUIT_Integrated_Payload,
  * SUIT_Integrated_Dependency,
  * $$SUIT_Envelope_Extensions,
  * (int => bstr)
}
```

Figure 2: SUIT Envelope CDDL.

The manifest is extended with a CEK verification parameter (called suit-cek-verification), see Figure 3. This parameter is optional and is utilized in environments where battery exhaustion attacks are a concern. Details about the CEK verification can be found in Section 7.

```
SUIT_Manifest = {
    suit-manifest-version         => 1,
    suit-manifest-sequence-number => uint,
    suit-common                   => bstr .cbor SUIT_Common,
    ? suit-reference-uri          => tstr,
    ? suit-cek-verification       => bstr,
    SUIT_Severable_Members_Choice,
    SUIT_Unseverable_Members,
    * $$SUIT_Manifest_Extensions,
}
```

Figure 3: SUIT Manifest CDDL.

5. AES Key Wrap

The AES Key Wrap (AES-KW) algorithm is described in RFC 3394 [RFC3394], and it can be used to encrypt a randomly generated content-encryption key (CEK) with a pre-shared key-encryption key (KEK). The COSE conventions for using AES-KW are specified in Section 12.2.1 of [RFC8152]. The encrypted CEK is carried in the COSE_recipient structure alongside the information needed for AES-KW. The COSE_recipient structure, which is a substructure of the COSE_Encrypt structure, contains the CEK encrypted by the KEK.

When the firmware image is encrypted for use by multiple recipients, there are three options. We use the following notation KEK(R1,S) is the KEK shared between recipient R1 and the sender S. Likewise, CEK(R1,S) is shared between R1 and S. If a single CEK or a single KEK is shared with all authorized recipients R by a given sender S in a certain context then we use CEK(*,S) or KEK(*,S), respectively. The notation ENC(plaintext, key) refers to the encryption of plaintext with a given key.

- If all authorized recipients have access to the KEK, a single COSE_recipient structure contains the encrypted CEK. This means KEK(*,S) ENC(CEK,KEK), and ENC(firmware,CEK).

- If recipients have different KEKs, then multiple COSE_recipient structures are included but only a single CEK is used. Each COSE_recipient structure contains the CEK encrypted with the KEKs appropriate for the recipient. In short, KEK_1(R1, S),..., KEK_n(Rn, S), ENC(CEK, KEK_i) for i=1 to n, and ENC(firmware,CEK). The benefit of this approach is that the firmware image is encrypted only once with a CEK while there is no sharing of the KEK accross recipients. Hence, authorized recipients still use their individual KEKs to decrypt the CEK and to subsequently obtain the plaintext firmware.

- The third option is to use different CEKs encrypted with KEKs of the authorized recipients. Assume there are KEK_1(R1, S),..., KEK_n(Rn, S), and for i=1 to n the following computations need to be made: ENC(CEK_i, KEK_i) and ENC(firmware,CEK_i). This approach is appropriate when no benefits can be gained from encrypting and transmitting firmware images only once. For example, firmware images may contain information unique to a device instance.

Note that the AES-KW algorithm, as defined in Section 2.2.3.1 of [RFC3394], does not have public parameters that vary on a per-invocation basis. Hence, the protected structure in the COSE_recipient is a byte string of zero length.

The COSE_Encrypt conveys information for encrypting the firmware image, which includes information like the algorithm and the IV, even though the firmware image is not embedded in the COSE_Encrypt.ciphertext itself since it conveyed as detached content.

The CDDL for the COSE_Encrypt_Tagged structure is shown in Figure 4.

```
COSE_Encrypt_Tagged = #6.96(COSE_Encrypt)

SUIT_Encryption_Info = COSE_Encrypt_Tagged

COSE_Encrypt = [
  protected   : bstr .cbor outer_header_map_protected,
  unprotected : outer_header_map_unprotected,
  ciphertext  : null,                  ; because of detached ciphertext
  recipients  : [ + COSE_recipient ]
]

outer_header_map_protected =
{
    1 => int,         ; algorithm identifier
  * label =values     ; extension point
}

outer_header_map_unprotected =
{
    5 => bstr,        ; IV
  * label =values     ; extension point
}

COSE_recipient = [
  protected   : bstr .size 0,
  unprotected : recipient_header_map,
  ciphertext  : bstr        ; CEK encrypted with KEK
]

recipient_header_map =
{
    1 => int,         ; algorithm identifier
    4 => bstr,        ; key identifier
  * label =values     ; extension point
}
```

Figure 4: CDDL for AES Key Wrap Encryption

The COSE specification requires a consistent byte stream for the authenticated data structure to be created, which is shown in Figure 5.

```
Enc_structure = [
  context : "Encrypt",
  protected : empty_or_serialized_map,
  external_aad : bstr
]
```

Figure 5: CDDL for Enc_structure Data Structure

As shown in Figure 4, there are two protected fields: one protected field in the COSE_Encrypt structure and a second one in the COSE_recipient structure. The 'protected' field in the Enc_structure, see Figure 5, refers to the content of the protected field from the COSE_Encrypt structure.

The value of the external_aad MUST be set to null.

The following example illustrates the use of the AES-KW algorithm with AES-128.

We use the following parameters in this example:

- IV: 0x26, 0x68, 0x23, 0x06, 0xd4, 0xfb, 0x28, 0xca, 0x01, 0xb4, 0x3b, 0x80

- KEK: "aaaaaaaaaaaaaaaa"

- KID: "kid-1"

- Plaintext Firmware: "This is a real firmware image."

- Firmware (hex): 546869732069732061207265616C206669726D77617265696D6167652E

The COSE_Encrypt structure, in hex format, is (with a line break inserted):

```
D8608443A10101A1054C26682306D4FB28CA01B43B80F68340A2012204456B69642D
315818AF09622B4F40F17930129D18D0CEA46F159C49E7F68B644D
```

The resulting COSE_Encrypt structure in a dignostic format is shown in Figure 6.

```
96(
    [
        / protected field with alg=AES-GCM-128 /
        h'A10101',
        {
           / unprotected field with iv /
           5: h'26682306D4FB28CA01B43B80'
        },
        / null because of detached ciphertext /
        null,
        [ / recipients array /
           h'', / protected field /
           {    / unprotected field /
              1: -3,            / alg=A128KW /
              4: h'6B69642D31'  / key id /
           },
           / CEK encrypted with KEK /
           h'AF09622B4F40F17930129D18D0CEA46F159C49E7F68B644D'
        ]
    ]
)
```

Figure 6: COSE_Encrypt Example for AES Key Wrap

The CEK, in hex format, was "4C805F1587D624ED5E0DBB7A7F7FA7EB" and the encrypted firmware (with a line feed added) was:

```
A8B6E61EF17FBAD1F1BF3235B3C64C06098EA512223260
F9425105F67F0FB6C92248AE289A025258F06C2AD70415
```

6. Hybrid Public-Key Encryption (HPKE)

Hybrid public-key encryption (HPKE) [I-D.irtf-cfrg-hpke] is a scheme that provides public key encryption of arbitrary-sized plaintexts given a recipient's public key.

For use with firmware encryption the scheme works as follows: HPKE, which internally utilizes a non-interactive ephemeral-static Diffie-Hellman exchange to derive a shared secret, is used to encrypt a CEK. This CEK is subsequently used to encrypt the firmware image. Hence, the plaintext passed to HPKE is the randomly generated CEK. The output of the HPKE SealBase function is therefore the encrypted CEK along with HPKE encapsulated key (i.e. the ephemeral ECDH public key).

Only the holder of recipient's private key can decapsulate the CEK to decrypt the firmware. Key generation in HPKE is influced by additional parameters, such as identity information.

This approach allows all recipients to use the same CEK to encrypt the firmware image, in case there are multiple recipients, to fulfill a requirement for the efficient distribution of firmware images using a multicast or broadcast protocol.

[I-D.ietf-cose-hpke] defines the use of HPKE with COSE.

An example of the COSE_Encrypt structure using the HPKE scheme is shown in Figure 7. It uses the following algorithm combination:

- AES-GCM-128 for encryption of the (detached) firmware image (at layer 0).
- AES-GCM-128 for encryption of the CEK in layer 1 as well as ECDH with NIST P-256 and HKDF-SHA256 as a Key Encapsulation Mechanism (KEM).

```
96_0([
    / protected header with alg=AES-GCM-128 /
    h'a10101',
    / unprotected header with nonce /
    {5: h'938b528516193cc7123ff037809f4c2a'},
    / detached ciphertext /
    null,
    / recipient structure /
    [
        / protected field with alg for HPKE /
        h'a1013863',
        / unprotected header /
        {
            / ephemeral public key with x / y coodinate /
            -1: h'a401022001215820a596f2ca8d159c04942308ca90
                  cfbfca65b108ca127df8fe191a063d00d7c5172258
                  20aef47a45d6d6c572e7bd1b9f3e69b50ad3875c68
                  f6da0caaa90c675df4162c39',
             /  kid for recipient static ECDH public key /
             4: h'6b69642d32',
        },
        / encrypted CEK /
        h'9aba6fa44e9b2cef9d646614dcda670dbdb31a3b9d37c7a
          65b099a8152533062',
    ],
])
```

Figure 7: COSE_Encrypt Example for HPKE

7. CEK Verification

The suit-cek-verification parameter contains a byte string resulting from the encryption of 8 bytes of 0xA5 using the CEK.

[[Editor's Note: Guidance about the selection of an IV needs to be added here.]]

8. Complete Examples

[[Editor's Note: Add examples for a complete manifest here (including a digital signature), multiple recipients, encryption of manifests (in comparison to firmware images).]]

9. Security Considerations

The algorithms described in this document assume that the party performing the firmware encryption

- shares a key-encryption key (KEK) with the firmware consumer (for use with the AES-Key Wrap scheme), or

- is in possession of the public key of the firmware consumer (for use with HPKE).

Both cases require some upfront communication interaction, which is not part of the SUIT manifest. This interaction is likely provided by an IoT device management solution, as described in [RFC9019].

For AES-Key Wrap to provide high security it is important that the KEK is of high entropy, and that implementations protect the KEK from disclosure. Compromise of the KEK may result in the disclosure of all key data protected with that KEK.

Since the CEK is randomly generated, it must be ensured that the guidelines for random number generations are followed, see [RFC8937].

In some cases third party companies analyse binaries for known security vulnerabilities. With encrypted firmware images this type of analysis is prevented. Consequently, these third party companies either need to be given access to the plaintext binary before encryption or they need to become authorized recipients of the encrypted firmware images. In either case, it is necessary to explicitly consider those third parties in the software supply chain when such a binary analysis is desired.

10. IANA Considerations

This document does not require any actions by IANA.

11. References

11.1. Normative References

[I-D.ietf-cose-hpke] Tschofenig, H., Housley, R., and B. Moran, "Use of Hybrid Public-Key Encryption (HPKE) with CBOR Object Signing and Encryption (COSE)", Work in Progress, Internet-Draft, draft-ietf-cose-hpke-00, 10 January 2022, <https://www.ietf.org/archive/id/draft-ietf-cose-hpke-00.txt>.

[I-D.ietf-suit-manifest] Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-16, 25 October 2021, <https://www.ietf.org/archive/id/draft-ietf-suit-manifest-16.txt>.

[I-D.irtf-cfrg-hpke] Barnes, R. L., Bhargavan, K., Lipp, B., and C. A. Wood, "Hybrid Public Key Encryption", Work in Progress, Internet-Draft, draft-irtf-cfrg-hpke-12, 2 September 2021, <https://www.ietf.org/archive/id/draft-irtf-cfrg-hpke-12.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/info/rfc3394>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, DOI 10.17487/RFC2630, June 1999, <https://www.rfc-editor.org/info/rfc2630>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC8937] Cremers, C., Garratt, L., Smyshlyaev, S., Sullivan, N., and C. Wood, "Randomness Improvements for Security Protocols", RFC 8937, DOI 10.17487/RFC8937, October 2020, <https://www.rfc-editor.org/info/rfc8937>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/info/rfc9019>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/info/rfc9124>.

Appendix A. Acknowledgements

We would like to thank Henk Birkholz for his feedback on the CDDL description in this document. Additionally, we would like to thank Michael Richardson and Carsten Bormann for their review feedback. Finally, we would like to thank Dick Brooks for making us aware of the challenges firmware encryption imposes on binary analysis.

Authors' Addresses

Hannes Tschofenig
Arm Limited

Email: hannes.tschofenig@arm.com

Russ Housley
Vigil Security, LLC

Email: housley@vigilsec.com

Brendan Moran
Arm Limited

Email: Brendan.Moran@arm.com