Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2015
M. Nottingham

M. Thomson
Mozilla
March 5, 2015

Encrypted Content-Encoding for HTTP
draft-nottingham-http-encryption-encoding-00

Abstract

This memo introduces a content-coding for HTTP that allows message payloads to be encrypted.

Table of Contents

1. Introduction

It is sometimes desirable to encrypt the contents of a HTTP message (request or response) in a persistent manner, so that when the payload is stored (e.g., with a HTTP PUT), only someone with the appropriate key can read it.

For example, it might be necessary to store a file on a server without exposing its contents to that server. Furthermore, that same file could be replicated to other servers (to make it more resistant to server or network failure), downloaded by clients (to make it available offline), etc. without exposing its contents.

These uses are not met by the use of TLS [RFC5246], since it only encrypts the channel between the client and server.

Message-based encryption formats - such as those that are described by [RFC4880], [RFC5652], [I-D.ietf-jose-json-web-encryption], and [XMLENC] - are not suited to stream processing, which is necessary for HTTP messages. While virtually any of these alternatives could be profiled and adapted to suit, the overhead and complexity that would introduce is sub-optimal.

This document specifies a content-coding [RFC7231]) for HTTP to serve these and other use cases.

This mechanism is likely only a small part of a larger design that uses content encryption. In particular, this document does not describe key management practices. How clients and servers acquire and identify keys will depend on the use case.

1.1. Notational Conventions

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].

2. The "aesgcm-128" HTTP content-coding

The "aesgcm-128" HTTP content-coding indicates that a payload has been encrypted using Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) as identified as AEAD_AES_128_GCM in [RFC5116], Section 5.1. The AEAD_AES_128_GCM algorithm uses a 128 bit content encryption key.

When this content-coding is in use, the Encryption header field Section 3 describes how encryption has been applied. The Encryption-Key header field Section 4 can be included to describe how the the content encryption key is derived or retrieved.

The "aesgcm-128" content-coding uses a single fixed set of encryption primitives. Cipher suite agility is achieved by defining a new content-coding scheme. This ensures that only the HTTP Accept-Encoding header field is necessary to negotiate the use of encryption.

The "aesgcm-128" content-coding uses a fixed record size. The resulting encoding is a series of fixed-size records, though the final record can contain any amount of data.

```
       +------+
       | data |         input of between rs-256
       +------+            and rs-1 octets
           |
           v
+-----+-----------+
| pad |   data    |     add padding to form plaintext
+-----+-----------+
         |
         v
+--------------------+
|    ciphertext      |  encrypt with AEAD_AES_128_GCM
+--------------------+     expands by 16 octets
```

The record size determines the length of each portion of plaintext that is enciphered. The record size defaults to 4096 octets, but can be changed using the "rs" parameter on the Encryption header field.

AEAD_AES_128_GCM expands ciphertext to be 16 octets longer than its input plaintext. Therefore, the length of each enciphered record is equal to the value of the "rs" parameter plus 16 octets. It is a fatal decryption error to have a remainder of 16 octets or less in size (though AEAD_AES_128_GCM permits input plaintext to be zero length, records always contain at least one padding octet).

Each record contains between 0 and 255 octets of padding, inserted into a record before the enciphered content. The length of the padding is stored in the first octet of the payload. All padding octets MUST be set to zero. It is a fatal decryption error to have a record with more padding than the record size.

The nonce used for each record is a 96-bit value containing the index of the current record in network byte order. Records are indexed starting at zero.

The additional data passed to the AEAD algorithm is a zero-length octet sequence.

Issue: Double check that having no AAD is safe.

3. The "Encryption" HTTP header field

The "Encryption" HTTP header field describes the encrypted content encoding(s) that have been applied to a message payload, and therefore how those content encoding(s) can be removed.

```
Encryption-val = #encryption_params
encryption_params = [ param *( ";" param ) ]
```

If the payload is encrypted more than once (as reflected by having multiple content-codings that imply encryption), each application of the content encoding is reflected in the Encryption header field, in the order in which they were applied.

The Encryption header MAY be omitted if the sender does not intend for the immediate recipient to be able to decrypt the message. Alternatively, the Encryption header field MAY be omitted if the sender intends for the recipient to acquire the header field by other means.

Servers processing PUT requests MUST persist the value of the Encryption header field, unless they remove the content-coding by decrypting the payload.

3.1. Encryption Header Field Parameters

The following parameters are used in determining the key that is used for encryption:

keyid: The "keyid" parameter contains a string that identifies the keying material that is used. The "keyid" parameter SHOULD be included, unless key identification is guaranteed by other means. The "keyid" parameter MUST be used if keying material is included in an Encryption-Key header field.

salt: The "salt" parameter contains a base64 URL-encoded octets that is used as salt in deriving a unique content encryption key (see Section 3.2). The "salt" parameter MUST be present, and MUST be exactly 16 octets long. The "salt" parameter MUST NOT be reused for two different messages that have the same content encryption key; generating a random nonce for each message ensures that reuse is highly unlikely.

rs: The "rs" parameter contains a positive decimal integer that describes the record size in octets. This value MUST be greater than 1. If the "rs" parameter is absent, the record size defaults to 4096 octets.

3.2. Content Encryption Key Derivation

In order to allow the reuse of keying material for multiple different messages, a content encryption key is derived for each message. This key is derived from the decoded value of the "s" parameter using the HMAC-based key derivation function (HKDF) described in [RFC5869] using the SHA-256 hash algorithm [FIPS180-2].

The decoded value of the "salt" parameter is the salt input to HKDF function. The keying material identified by the "keyid" parameter is the input keying material (IKM) to HKDF. Input keying material can either be prearranged, or can be described using the Encryption-Key header field Section 4. The first step of HKDF is therefore:

```
PRK = HMAC-SHA-256(salt, IKM)
```

AEAD_AES_128_GCM requires 16 octets (128 bits) of key, so the length (L) parameter of HKDF is 16. The info parameter is set to the ASCII-encoded string "Content-Encoding: aesgcm128". The second step of HKDF can therefore be simplified to the first 16 octets of a single HMAC:

```
OKM = HMAC-SHA-256(PRK, "Content-Encoding: aesgcm128" || 0x01)
```

4. Encryption-Key Header Field

An Encryption-Key header field can be used to describe the input keying material used in the Encryption header field.

```
Encryption-Key-val = #encryption_key_params
encryption_key_params = [ param *( ";" param ) ]
```

keyid: The "keyid" parameter corresponds to the "keyid" parameter in the Encryption header field.

key: The "key" parameter contains the URL-safe base64 [RFC4648] octets of the input keying material.

dh: The "dh" parameter contains an ephemeral Diffie-Hellman share. This form of the header field can be used to encrypt content for a specific recipient.

The input keying material used by the content-encoding key derivation (see Section 3.2) can be determined based on the information in the Encryption-Key header field. The method for key derivation depends on the parameters that are present in the header field.

Note that different methods for determining input keying materal will produce different amounts of data. The HKDF process ensures that the final content encryption key is the necessary size.

Alternative methods for determining input keying material MAY be defined by specifications that use this content-encoding.

4.1. Explicit Key

The "key" parameter is decoded and used directly if present. The "key" parameter MUST decode to exactly 16 octets in order to be used as input keying material for "aesgcm128" content encoding.

Other key determination parameters can be ignored if the "key" parameter is present.

4.2. Diffie-Hellman

The "dh" parameter is included to describe a Diffie-Hellman share, either modp (or finite field) Diffie-Hellman [DH] or elliptic curve Diffie-Hellman (ECDH) [RFC4492].

This share is combined with other information at the recipient to determine the HKDF input keying material. In order for the exchange to be successful, the following information MUST be established out of band:

- o Which Diffie-Hellman form is used.
- o The modp group or elliptic curve that will be used.
- o The format of the ephemeral public share that is included in the "dh" parameter. For instance, using ECDH both parties need to agree whether this is an uncompressed or compressed point.

In addition to identifying which content-encoding this input keying material is used for, the "keyid" parameter is used to identify this additional information at the receiver.

The intended recipient recovers their private key and are then able to generate a shared secret using the appropriate Diffie-Hellman process.

Specifications that rely on an Diffie-Hellman exchange for determining input keying material MUST either specify the parameters for Diffie-Hellman (group parameters, or curves and point format) that are used, or describe how those parameters are negotiated between sender and receiver.

5. Examples

5.1. Successful GET Response

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Encoding: aesgcm-128
Connection: close
Encryption: keyid="http://example.org/bob/keys/123";
            salt="XZwpw6o37R-6qoZjw6KwAw"

[encrypted payload]
```

Here, a successful HTTP GET response has been encrypted using a key that is identified by a URI.

Note that the media type has been changed to "application/octet-stream" to avoid exposing information about the content.

5.2. Encryption and Compression

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Encoding: aesgcm-128, gzip
Transfer-Encoding: chunked
Encryption: keyid="mailto:me@example.com";
            salt="m2hJ_NttRtFyUiMRPwfpHA"

[encrypted payload]
```

5.3. Encryption with More Than One Key

```
PUT /thing HTTP/1.1
Host: storage.example.com
Content-Type: application/http
Content-Encoding: aesgcm-128, aesgcm-128
Content-Length: 1234
Encryption: keyid="mailto:me@example.com";
            salt="NfzOeuV5USPRA-n_9s1Lag",
            keyid="http://example.org/bob/keys/123";
            salt="bDMSGoc2uobK_IhavSHsHA"; rs=1200

[encrypted payload]
```

Here, a PUT request has been encrypted with two keys; both will be necessary to read the content. The outer layer of encryption uses a 1200 octet record size.

5.4. Encryption with Explicit Key

```
HTTP/1.1 200 OK
Content-Length: 31
Content-Encoding: aesgcm-128
Encryption: keyid="a1"; salt="owIfQR647esVfrzCW_i9GQ"
Encryption-Key: keyid="a1"; key="JcqK-OLkJZlJ3sJJWstJCA"

LwTC-fwdKh8de0smD2jfzHodb1EYbuuTNpcYXLW257Q
```

This example shows the string "I am the walrus" encrypted using an explicit key. The content body contains a single record only and is shown here encoded in URL-safe base64 for presentation reasons only.

5.5. Diffie-Hellman Encryption

```
HTTP/1.1 200 OK
Content-Length: 31
Content-Encoding: aesgcm-128
Encryption: keyid="dhkey"; salt="XYFSCgMVjc45IMfLOcMfiw"
Encryption-Key: keyid="dhkey";
                dh="BELKqvZ7n3p5C9_ipP_6X9DBNAGuJujSN7YWbtcGZMMH
                    3urZM-zlii3mGGCMjlqR-yWwiPlMdKRdOL8gQSdHw8E"

P6ikHE_wyKnYHXxLswvuFBO3JJOZpM1Bg3KikQEmczU
```

This example shows the same string, "I am the walrus", encrypted using ECDH over the P-256 curve [FIPS186]. The content body is shown here encoded in URL-safe base64 for presentation reasons only.

The receiver (in this case, the HTTP client) uses the key identified by the string "dhkey" and the sender (the server) uses a key pair for which the public share is included in the "dh" parameter above. The keys shown below use uncompressed points [X.692] encoded using URL-safe base64. Line wrapping is added for presentation purposes only.

```
Receiver:
   private key: QjGwenE3vCg8Eajo-PukGgUkYq8Vu-SQn04Cc9DR-YA
   public key: BBM3pYS4nXG6bQYnZbGDY7l6CVrQTZ-1u00h7XV6A_TD
               v7mXvv5k29uoLid8SdDycw341PJTW4hNCe2FNysN52U
Sender:
   private key: wlC-qzKBWO6jYq32nlD0ZZVsI5jGVBC1gN7zkXjaPks
   public key: <the value of the "dh" parameter>
```

6. IANA Considerations

6.1. The "aesgcm-128" HTTP content-coding

This memo registers the "encrypted" HTTP content-coding in the HTTP Content Codings Registry, as detailed in Section 2.

- o Name: aesgcm-128
- o Description: AES-GCM encryption with a 128-bit key
- o Reference [this specification]

6.2. Encryption Header Fields

This memo registers the "Encryption" HTTP header field in the Permanent Message Header Registry, as detailed in Section 3.

- o Field name: Encryption
- o Protocol: HTTP
- o Status: Standard
- o Reference: [this specification]
- o Notes:

This memo registers the "Encryption-Key" HTTP header field in the Permanent Message Header Registry, as detailed in Section 4.

- o Field name: Encryption-Key
- o Protocol: HTTP
- o Status: Standard
- o Reference: [this specification]
- o Notes:

6.3. The HTTP Encryption Parameter Registry

This memo establishes a registry for parameters used by the "Encryption" header field under the "Hypertext Transfer Protocol (HTTP) Parameters" grouping. The "Hypertext Transfer Protocol (HTTP) Encryption Parameters" operates under an "Specification Required" policy [RFC5226].

Entries in this registry are expected to include the following information:

- o Parameter Name: The name of the parameter.
- o Purpose: A brief description of the purpose of the parameter.
- o Reference: A reference to a specification that defines the semantics of the parameter.

The initial contents of this registry are:

6.3.1. keyid

- o Parameter Name: keyid
- o Purpose: Identify the key that is in use.
- o Reference: [this document]

6.3.2. salt

- o Parameter Name: salt
- o Purpose: Provide a source of entropy for derivation of the content encryption key. This value is mandatory.
- o Reference: [this document]

6.3.3. rs

- o Parameter Name: rs
- o Purpose: The size of the encrypted records.
- o Reference: [this document]

6.4. The HTTP Encryption-Key Parameter Registry

This memo establishes a registry for parameters used by the "Encryption-Key" header field under the "Hypertext Transfer Protocol (HTTP) Parameters" grouping. The "Hypertext Transfer Protocol (HTTP) Encryption Parameters" operates under an "Specification Required" policy [RFC5226].

Entries in this registry are expected to include the following information:

- Parameter Name: The name of the parameter.

- Purpose: A brief description of the purpose of the parameter.

- Reference: A reference to a specification that defines the semantics of the parameter.

The initial contents of this registry are:

6.4.1. keyid

- Parameter Name: keyid

- Purpose: Identify the key that is in use.

- Reference: [this document]

6.4.2. key

- Parameter Name: key

- Purpose: Provide an explicit key.

- Reference: [this document]

6.4.3. dh

- Parameter Name: dh

- Purpose: Carry a modp or elliptic curve Diffie-Hellman share used to derive a key.

- Reference: [this document]

7. Security Considerations

This mechanism assumes the presence of a key management framework that is used to manage the distribution of keys between valid senders and receivers. Defining key management is part of composing this mechanism into a larger application, protocol, or framework.

Implementation of cryptography - and key management in particular - can be difficult. For instance, implementations need to account for the potential for exposing keying material on side channels, such as might be exposed by the time it takes to perform a given operation. The requirements for a good implementation of cryptographic algorithms can change over time.

7.1. Key and Nonce Reuse

Encrypting different plaintext with the same content encryption key and nonce in AES-GCM is not safe [RFC5116]. The scheme defined here relies on the uniqueness of the "nonce" parameter to ensure that the content encryption key is different for every message.

If a key and nonce are reused, this could expose the content encryption key and it makes message modification trivial. If the same key is used for multiple messages, then the nonce parameter MUST be unique for each. An implementation SHOULD generate a random nonce parameter for every message, though using a counter could achieve the desired result.

7.2. Content Integrity

This mechanism only provides content origin authentication. The authentication tag only ensures that those with access to the content encryption key produce a message that will be accepted as valid.

Any entity with the content encryption key can therefore produce content that will be accepted as valid. This includes all recipients of the same message.

Furthermore, any entity that is able to modify both the Encryption header field and the message payload can replace messages. Without the content encryption key however, modifications to or replacement of parts of a message are not possible.

7.3. Leaking Information in Headers

Because "encrypted" only operates upon the message payload, any information exposed in headers is visible to anyone who can read the message.

For example, the Content-Type header can leak information about the message payload.

There are a number of strategies available to mitigate this threat, depending upon the application's threat model and the users' tolerance for leaked information:

1. Determine that it is not an issue. For example, if it is expected that all content stored will be "application/json", or another very common media type, exposing the Content-Type header could be an acceptable risk.

2. If it is considered sensitive information and it is possible to determine it through other means (e.g., out of band, using hints in other representations, etc.), omit the relevant headers, and/ or normalize them. In the case of Content-Type, this could be accomplished by always sending Content-Type: application/octet-stream (the most generic media type).

3. If it is considered sensitive information and it is not possible to convey it elsewhere, encapsulate the HTTP message using the application/http media type [RFC7230], encrypting that as the payload of the "outer" message.

7.4. Poisoning Storage

This mechanism only offers encryption of content; it does not perform authentication or authorization, which still needs to be performed (e.g., by HTTP authentication [RFC7235]).

This is especially relevant when a HTTP PUT request is accepted by a server; if the request is unauthenticated, it becomes possible for a third party to deny service and/or poison the store.

7.5. Sizing and Timing Attacks

Applications using this mechanism need to be aware that the size of encrypted messages, as well as their timing, HTTP methods, URIs and so on, may leak sensitive information.

This risk can be mitigated through the use of the padding that this mechanism provides. Alternatively, splitting up content into segments and storing the separately might reduce exposure. HTTP/2 [I-D.ietf-httpbis-http2] combined with TLS [RFC5246] might be used to hide the size of individual messages.

8. References

8.1. Normative References

[DH] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, V.IT-22 n.6 , June 1977.

[FIPS180-2] Department of Commerce, National., "NIST FIPS 180-2, Secure Hash Standard", August 2002.

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

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

8.2. Informative References

[FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.

[I-D.ietf-httpbis-http2] Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-17 (work in progress), February 2015.

[I-D.ietf-jose-json-web-encryption] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-40 (work in progress), January 2015.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[X.692] ANSI, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 1998. , n.d..

[XMLENC] Eastlake, D., Reagle, J., Imamura, T., Dillaway, B., and E. Simon, "XML Encryption Syntax and Processing", W3C REC , December 2002, <http://www.w3.org/TR/xmlenc-core/>.

Appendix A. Acknowledgements

The following people provided valuable feedback and suggestions: Richard Barnes, Stephen Farrell, Eric Rescorla, and Jim Schaad.

Authors' Addresses

Mark Nottingham

Email: mnot@mnot.net
URI: http://www.mnot.net/

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com