Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 12, 2017

M. Thomson
Mozilla

October 9, 2016

Message Encryption for Web Push
draft-ietf-webpush-encryption-04

Abstract

A message encryption scheme is described for the Web Push protocol. This scheme provides confidentiality and integrity for messages sent from an Application Server to a User Agent.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

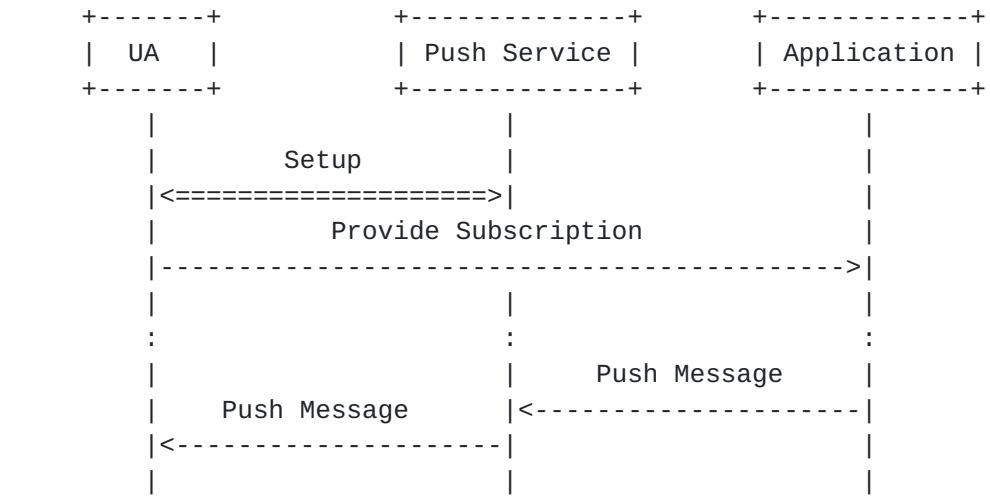
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 12, 2017.

Table of Contents

1. Introduction

The Web Push protocol [I-D.ietf-webpush-protocol] is an intermediated protocol by necessity. Messages from an Application Server are delivered to a User Agent via a Push Service.



This document describes how messages sent using this protocol can be secured against inspection, modification and falsification by a Push Service.

Web Push messages are the payload of an HTTP message [RFC7230]. These messages are encrypted using an encrypted content encoding [I-D.ietf-httpbis-encryption-encoding]. This document describes how this content encoding is applied and describes a recommended key management scheme.

For efficiency reasons, multiple users of Web Push often share a central agent that aggregates push functionality. This agent can enforce the use of this encryption scheme by applications that use push messaging. An agent that only delivers messages that are properly encrypted strongly encourages the end-to-end protection of messages.

A web browser that implements the Web Push API [API] can enforce the use of encryption by forwarding only those messages that were properly encrypted.

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting, when they are capitalized, they have the special meaning described in [RFC2119].

2. Push Message Encryption Overview

Encrypting a push message uses elliptic-curve Diffie-Hellman (ECDH) [ECDH] on the P-256 curve [FIPS186] to establish a shared secret (see Section 3.1) and a symmetric secret for authentication (see Section 3.2).

A User Agent generates an ECDH key pair and authentication secret that it associates with each subscription it creates. The ECDH public key and the authentication secret are sent to the Application Server with other details of the push subscription.

When sending a message, an Application Server generates an ECDH key pair and a random salt. The ECDH public key is encoded into the "dh" parameter of the Crypto-Key header field; the salt is encoded into the "salt" parameter of the Encryption header field. The ECDH key pair can be discarded after encrypting the message.

The content of the push message is encrypted or decrypted using a content encryption key and nonce that is derived using all of these inputs and the process described in Section 3.

2.1. Key and Secret Distribution

The application using the subscription distributes the subscription public key and authentication secret to an authorized Application Server. This could be sent along with other subscription information that is provided by the User Agent, such as the push subscription URI.

An application MUST use an authenticated, confidentiality protected communications medium for this purpose. In addition to the reasons described in [I-D.ietf-webpush-protocol], this ensures that the authentication secret is not revealed to unauthorized entities, which can be used to generate push messages that will be accepted by the User Agent.

Most applications that use push messaging have a pre-existing relationship with an Application Server. Any existing communication mechanism that is authenticated and provides confidentiality and integrity, such as HTTPS [RFC2818], is sufficient.

3. Push Message Encryption

Push message encryption happens in four phases:

- The input keying material used for deriving the content encryption keys used for the push message is derived using elliptic-curve Diffie-Hellman [ECDH] (Section 3.1).

- This is then combined with the application secret to produce the input keying material used in [I-D.ietf-httpbis-encryption-encoding] (Section 3.3).

- A content encryption key and nonce are derived using the process in [I-D.ietf-httpbis-encryption-encoding] with an expanded context string (Section 3.4).

- Encryption or decryption follows according to [I-D.ietf-httpbis-encryption-encoding].

The key derivation process is summarized in Section 3.5. Restrictions on the use of the encrypted content coding are described in Section 4.

3.1. Diffie-Hellman Key Agreement

For each new subscription that the User Agent generates for an Application, it also generates a P-256 [FIPS186] key pair for use in elliptic-curve Diffie-Hellman (ECDH) [ECDH].

When sending a push message, the Application Server also generates a new ECDH key pair on the same P-256 curve.

The ECDH public key for the Application Server is included in the "dh" parameter of the Crypto-Key header field (see Section 6). The uncompressed point form defined in [X9.62] (that is, a 65 octet sequence that starts with a 0x04 octet) is encoded using base64url [RFC7515] to produce the "dh" parameter value.

An Application combines its ECDH private key with the public key provided by the User Agent using the process described in [ECDH]; on receipt of the push message, a User Agent combines its private key with the public key provided by the Application Server in the "dh" parameter in the same way. These operations produce the same value for the ECDH shared secret.

3.2. Push Message Authentication

To ensure that push messages are correctly authenticated, a symmetric authentication secret is added to the information generated by a User Agent. The authentication secret is mixed into the key derivation process described in [I-D.ietf-httpbis-encryption-encoding].

A User Agent MUST generate and provide a hard to guess sequence of 16 octets that is used for authentication of push messages. This SHOULD be generated by a cryptographically strong random number generator [RFC4086].

3.3. Combining Shared and Authentication Secrets

The shared secret produced by ECDH is combined with the authentication secret using HMAC-based key derivation function (HKDF) described in [RFC5869]. This produces the input keying material used by [I-D.ietf-httpbis-encryption-encoding].

The HKDF function uses SHA-256 hash algorithm [FIPS180-4] with the following inputs:

salt: the authentication secret

IKM: the shared secret derived using ECDH

info: the ASCII-encoded string "Content-Encoding: auth" with a terminal zero octet

L: 32 octets (i.e., the output is the length of the underlying SHA-256 HMAC function output)

3.4. Key Derivation Context

The derivation of the content encryption key and nonce uses an additional context string.

The context is comprised of a label of "P-256" encoded in ASCII (that is, the octet sequence 0x50, 0x2d, 0x32, 0x35, 0x36), a zero-valued octet, the length of the User Agent public key (65 octets) encoded as a two octet unsigned integer in network byte order, the User Agent public key, the length of the Application Server public key (65 octets), and the Application Server public key.

```
context = label || 0x00 ||
            length(ua_public) || ua_public ||
            length(as_public) || as_public
```

3.5. Encryption Summary

This results in a the final content encryption key and nonce generation using the following sequence, which is shown here in pseudocode with HKDF expanded into separate discrete steps using HMAC with SHA-256:

```
-- For a User Agent:
ecdh_secret = ECDH(ua_private, as_public)
auth_secret = random(16)

-- For an Application Server:
ecdh_secret = ECDH(as_private, ua_public)
auth_secret = <from User Agent>

-- For both:
auth_info = "Content-Encoding: auth" || 0x00
PRK_combine = HMAC-SHA-256(auth_secret, ecdh_secret)
IKM = HMAC-SHA-256(PRK_combine, auth_info || 0x01)
context = "P-256" || 0x00 ||
          0x00 || 0x41 || ua_public ||
          0x00 || 0x41 || as_public
salt = random(16)
PRK = HMAC-SHA-256(salt, IKM)
cek_info = "Content-Encoding: aesgcm" || 0x00 || context
CEK = HMAC-SHA-256(PRK, cek_info || 0x01)[0..15]
nonce_info = "Content-Encoding: nonce" || 0x00 || context
NONCE = HMAC-SHA-256(PRK, nonce_info || 0x01)[0..11]
```

Note that this omits the exclusive OR of the final nonce with the record sequence number, since push messages contain only a single

record (see Section 4) and the sequence number of the first record is zero.

4. Restrictions on Use of "aesgcm" Content Coding

An Application Server MUST encrypt a push message with a single record. This allows for a minimal receiver implementation that handles a single record. If the message is 4096 octets or longer, the "rs" parameter MUST be set to a value that is longer than the encrypted push message length.

A push service is not required to support more than 4096 octets of payload body (see Section 7.2 of [I-D.ietf-webpush-protocol]), which equates to 4077 octets of cleartext, so the "rs" parameter can be omitted for messages that fit within this limit.

An Application Server MUST NOT use other content encodings for push messages. In particular, content encodings that compress could result in leaking of push message contents. The Content-Encoding header field therefore has exactly one value, which is "aesgcm". Multiple "aesgcm" values are not permitted.

An Application Server MUST include exactly one entry in the Encryption field, and at most one entry having a "dh" parameter in the Crypto-Key field. This allows the "keyid" parameter to be omitted from both header fields.

An Application Server MUST NOT include an "aesgcm" parameter in the Encryption header field.

A User Agent is not required to support multiple records. A User Agent MAY ignore the "rs" parameter. If a record size is size is present, but unchecked, decryption will fail with high probability for all valid cases. Decryption will also succeed if the push message contains a single record from a longer truncated message. Given that an Application Server is prohibited from generating such a message, this is not considered a serious risk.

5. Push Message Encryption Example

The following example shows a push message being sent to a push service.

```
POST /push/JzLQ3raZJfFBR0aqvOMsLrt54w4rJUsV HTTP/1.1
Host: push.example.net
TTL: 10
Content-Length: 33
Content-Encoding: aesgcm
Encryption: salt="lngarbyKfMoi9Z75xYXmkg"
Crypto-Key: dh="BNoRDbb84JGm8g5Z5CFxurSqsXWJ11ItfXEWYVLE85Y7
                CYkDjXsIEc4aqxYaQ1G8BqkXCJ6DPpDrWtdWj_mugHU"

6nqAQUME8hNqw5J3kl8cpVVJylXKYqZOeseZG8UueKpA
```

This example shows the ASCII encoded string, "I am the walrus". The content body is shown here encoded in URL-safe base64url for presentation reasons only. Line wrapping of the "dh" parameter is added for presentation purposes.

Since there is no ambiguity about which keys are being used, the "keyid" parameter is omitted from both the Encryption and Crypto-Key header fields. The keys shown below use uncompressed points [X9.62] encoded using base64url.

```
Authentication Secret: R29vIGdvbyBnJyBqb29iIQ
Receiver:
   private key: 9FWl15_QUQAWDaD3k3l50ZBZQJ4au27F1V4F0uLSD_M
   public key: BCEkBjzL8Z3C-oi2Q7oE5t2Np-p7osjGLg93qUP0wvqR
               T21EEWyf0cQDQcakQMqz4hQKYOQ3il2nNZct4HgAUQU
Sender:
   private key: nCScek-QpEjmOOlT-rQ38nZzvdPlqa00Zy0i6m2OJvY
   public key: <the value of the "dh" parameter>
```

The sender's private key used in this example is "nCScek-QpEjmOOlT-rQ38nZzvdPlqa00Zy0i6m2OJvY". Intermediate values for this example are included in Appendix A.

6. IANA Considerations

This document defines the "dh" parameter for the Crypto-Key header field in the "Hypertext Transfer Protocol (HTTP) Crypto-Key Parameters" registry defined in [I-D.ietf-httpbis-encryption-encoding].

- Parameter Name: dh
- Purpose: The "dh" parameter contains a Diffie-Hellman share which is used to derive the input keying material used in "aesgcm" content coding.
- Reference: this document.

7. Security Considerations

The security considerations of [I-D.ietf-httpbis-encryption-encoding] describe the limitations of the content encoding. In particular, any HTTP header fields are not protected by the content encoding scheme. A User Agent MUST consider HTTP header fields to have come from the Push Service. An application on the User Agent that uses information from header fields to alter their processing of a push message is exposed to a risk of attack by the Push Service.

The timing and length of communication cannot be hidden from the Push Service. While an outside observer might see individual messages intermixed with each other, the Push Service will see what Application Server is talking to which User Agent, and the subscription that is used. Additionally, the length of messages could be revealed unless the padding provided by the content encoding scheme is used to obscure length.

8. References

8.1. Normative References

[ECDH] SECG, "Elliptic Curve Cryptography", SEC 1 , 2000, <http://www.secg.org/>.

[FIPS180-4] Department of Commerce, National., "NIST FIPS 180-4, Secure Hash Standard", March 2012, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

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

[I-D.ietf-httpbis-encryption-encoding] Thomson, M., "Encrypted Content-Encoding for HTTP", draft-ietf-httpbis-encryption-encoding-02 (work in progress), June 2016.

[I-D.ietf-webpush-protocol] Thomson, M., Damaggio, E., and B. Raymor, "Generic Event Delivery Using HTTP Push", draft-ietf-webpush-protocol-10 (work in progress), September 2016.

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

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

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

8.2. Informative References

[API] van Ouwerkerk, M. and M. Thomson, "Web Push API", 2015, <https://w3c.github.io/push-api/>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Appendix A. Intermediate Values for Encryption

The intermediate values calculated for the example in Section 5 are shown here. The following are inputs to the calculation:

Plaintext: SSBhbSB0aGUgd2FscnVz

Application Server public key (as_public):
BNoRDbb84JGm8g5Z5CFxurSqsXWJ11ItfXEWYVLE85Y7
CYkDjXsIEc4aqxYaQ1G8BqkXCJ6DPpDrWtdWj_mugHU

Application Server private key (as_private): nCScek-QpEjmOOlT-rQ38nZ
zvdPlqa00Zy0i6m2OJvY

User Agent public key (ua_public): BCEkBjzL8Z3C-oi2Q7oE5t2Np-
p7osjGLg93qUP0wvqR T21EEWyf0cQDQcakQMqz4hQKYOQ3il2nNZct4HgAUQU

User Agent private key (ua_private):
9FWl15_QUQAWDaD3k3l50ZBZQJ4au27F1V4F0uLSD_M

Salt: lngarbyKfMoi9Z75xYXmkg

Authentication secret (auth_secret): R29vIGdvbyBnJyBqb29iIQ

Note that knowledge of just one of the private keys is necessary.
The Application Server randomly generates the salt value, whereas
salt is input to the receiver.

This produces the following intermediate values:

Shared secret (ecdh_secret): RNjC-
NVW4BGJbxWPW7G2mowsLeDa53LYKYm4-NOQ6Y

Input keying material (IKM): EhpZec37Ptm4IRD5-jtZ0q6r1iK5vYmY1tZwtN8
fbZY

Context for content encryption key derivation:
Q29udGVudC1FbmNvZGluZzogYWVzZ2NtAFAtMjU2AABB BCEkBjzL8Z3C-
oi2Q7oE5t2Np-p7osjGLg93qUP0wvqR
T21EEWyf0cQDQcakQMqz4hQKYOQ3il2nNZct4HgAUQUA
QQTaEQ22_OCRpvIOWeQhcbq0qrF1iddSLX1xFmFSxPOW
OwmJA417CBHOGqsWGkNRvAapFwiegz6Q61rXVo_5roB1

Content encryption key (CEK): AN2-xhvFWeYh5z0fcDu0Ww

Context for nonce derivation: Q29udGVudC1FbmNvZGluZzogbm9uY2UAUC0yNT
YAAEEE ISQGPMvxncL6iLZDugTm3Y2n6nuiyMYuD3epQ_TC-pFP
bUQRbJ_RxANBxqRAyrPiFApg5DeKXac1ly3geABRBQBB
BNoRDbb84JGm8g5Z5CFxurSqsXWJ11ItfXEWYVLE85Y7
CYkDjXsIEc4aqxYaQ1G8BqkXCJ6DPpDrWtdWj_mugHU

Base nonce: JY1Okw5rw1Drkg9J

When the CEK and nonce are used with AES GCM and the padded plaintext
of AABJIGFtIHRoZSB3YWxydXM, the final ciphertext is
6nqAQUME8hNqw5J3kl8cpVVJylXKYqZOeseZG8UueKpA, as shown in the
example.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com