Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

M. Thomson
Mozilla

October 31, 2016

Message Encryption for Web Push
draft-ietf-webpush-encryption-05

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/.

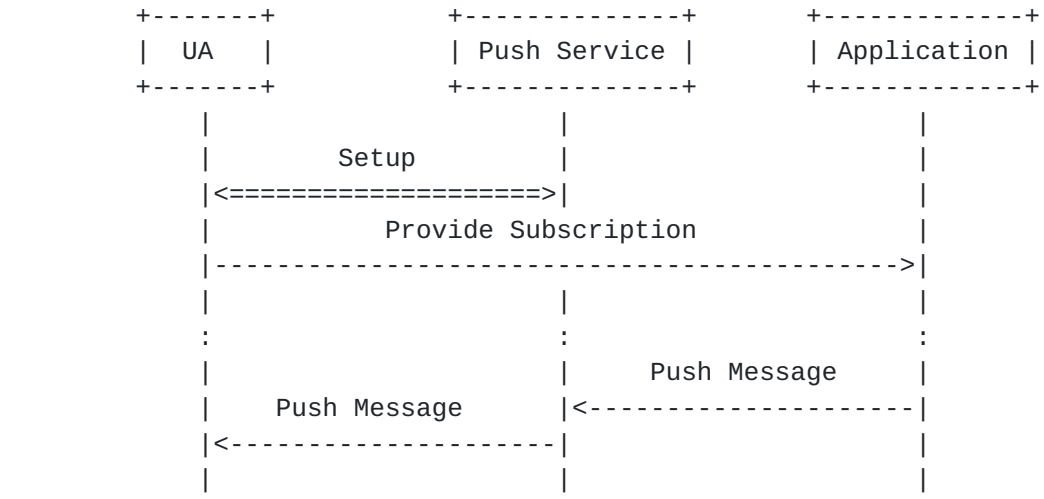
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 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 message payload. 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:

- A shared secret is derived using elliptic-curve Diffie-Hellman [ECDH] (Section 3.1).
- The shared secret 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].
- Encryption or decryption follows according to [I-D.ietf-httpbis-encryption-encoding].

The key derivation process is summarized in Section 3.4. 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 shown in Section 3.3.

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 concatenation of the ASCII-encoded string "WebPush: info", a zero octet, the X9.62 encoding of the User Agent ECDH public key, and X9.62 encoding of the Application Server ECDH public key; that is

key_info = "WebPush: info" || 0x00 || ua_public || as_public

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

3.4. 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:
PRK_key = HMAC-SHA-256(auth_secret, ecdh_secret)
key_info = "WebPush: info" || 0x00 || ua_public || as_public
IKM = HMAC-SHA-256(PRK_cek, key_info || 0x01)

salt = random(16)
PRK = HMAC-SHA-256(salt, IKM)
cek_info = "Content-Encoding: aes128gcm" || 0x00
CEK = HMAC-SHA-256(PRK, cek_info || 0x01)[0..15]
nonce_info = "Content-Encoding: nonce" || 0x00
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 "aes128gcm" 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. An application server MUST set the "rs" parameter in the "aes128gcm" content coding header to a size that is greater than the length of the plaintext, plus any padding (which is at least 2 octets).

A push message MUST include a zero length "keyid" parameter in the content coding header. This allows implementations to ignore the first 21 octets of a push message.

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 at most 4059 octets of cleartext.

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 "aes128gcm". Multiple "aes128gcm" values are not permitted.

An Application Server MUST include exactly one "aes128gcm" content coding, and at most one entry in the Crypto-Key field. This allows the "keyid" parameter to be omitted.

An Application Server MUST NOT include an "aes128gcm" parameter in the Crypto-Key header field.

A User Agent is not required to support multiple records. A User Agent MAY ignore the "rs" field and assume that the "keyid" field is empty. If a record size is unchecked, decryption will fail with high probability for all valid cases. However, 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: aes128gcm
Crypto-Key: dh=BP4z9KsN6nGRTbVYI_c7VJSPQTBtkgcy27mlmlMoZIIg
               Dll6e3vCYLocInmYWAmS6TlzAC8wEqKK6PBru3jl7A8

DGv6ra1nlYgDCS1FRnbzlwAAxowAIg1VvoJvrVBFhclGlx4G2FuProCVzJY04Lg5
vUP2LeswtWoBGHGoYXUzAwuxQGRGxoNbh8BROK3gmJ0
```

This example shows the ASCII encoded string, "When I grow up, I want to be a watermelon". The content body is shown here with line wrapping and URL-safe base64url encoding to meet presentation constraints. Similarly, the "dh" parameter wrapped to meet line length constraints.

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: BTBZMqHH6r4Tts7J_aSIgg
   Receiver:
      private key: q1dXpw3UpT5VOmu_cf_v6ih07Aems3njxI-JWgLcM94
      public key: BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-JvLexhqUzORcx
                  aOzi6-AYWXvTBHm4bjyPjs7Vd8pZGH6SRpkNtoIAiw4
   Sender:
      private key: yfWPiYE-n46HLnH0KqZOF1fJJU3MYrct3AELtAQ-oRw
      public key: <the value of the "dh" parameter>
```

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 "aes128gcm" 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-03 (work in progress), October 2016.

[I-D.ietf-webpush-protocol] Thomson, M., Damaggio, E., and B. Raymor, "Generic Event Delivery Using HTTP Push", draft-ietf-webpush-protocol-12 (work in progress), October 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: V2hlbiBJIGdyb3cgdXAsIEkgd2FudCB0byBiZSBhIHdhdGVybWVsb24

Application Server public key (as_public):
BP4z9KsN6nGRTbVYI_c7VJSPQTBtkgcy27mlmlMoZIIg
Dll6e3vCYLocInmYWAmS6TlzAC8wEqKK6PBru3jl7A8

Application Server private key (as_private): yfWPiYE-n46HLnH0KqZOF1f
JJU3MYrct3AELtAQ-oRw

User Agent public key (ua_public): BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-
JvLexhqUzORcx aOzi6-AYWXvTBHm4bjyPjs7Vd8pZGH6SRpkNtoIAiw4

User Agent private key (ua_private):
q1dXpw3UpT5VOmu_cf_v6ih07Aems3njxI-JWgLcM94

Salt: DGv6ra1nlYgDCS1FRnbzlw

Authentication secret (auth_secret): BTBZMqHH6r4Tts7J_aSIgg

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 ECDH secret (ecdh_secret): kyrL1jIIOHEzg3sM2ZWRHDRB62YACZhhSl knJ672kSs

Pseudo-random key for key combining (PRK_key): Snr3JMxaHVDXHWJn5wdC52WjpCtd2EIEGBykDcZW32k

Info for key combining (key_info): V2ViUHVzaDogaW5mbwAE_jP0qw3qcZFNt Vgj9ztUlI9 BMG2SBzLbuaWaUyhkgiAOWXp7e8JguhwieZhYCZLpOX MALzASooro8Gu7eOXsDwQlcbK-zf3jYFUarx7Q9M02b BHOvlVfiby3sYalMzkXMWjs4uvgGFl70wR5uG48j47O 1XfKWRh-kkaZDbaCAIsO

Input keying material for content encryption key derivation (IKM): dTQXtQpktdp6UQb29SUBcO5igFtC9WsXlhlNr2jRkkY

PRK for content encryption (PRK): BEhmz5JYdOXMsFJf_WDU8fJlOURaExoUoF uaGU86Fuc

Info for content encryption key derivation (cek_info): Q29udGVudC1FbmNvZGluZzogYWVzZ2NtMTI4AA

Content encryption key (CEK): wgJKGPLNgnI3CKy09z19Qw

Info for content encryption nonce derivation (nonce_info): Q29udGVudC1FbmNvZGluZzogbm9uY2UA

Nonce (NONCE): w5SniqvyjVui9OoV

The salt and a record size of 4096 produce a 21 octet header of DGv6ra1nlYgDCS1FRnbzlwAAxowA.

The push message plaintext is padded to produce AABXaGVuIEkgZ3JvdyB1cCwgSSB3YW50IHRvIGJl IGEgd2F0ZXJtZWxvbg. The plaintext is then encrypted with AES-GCM, which emits ciphertext of Ig1VvoJvrVBFhclGlx4G2FuProCVzJY04Lg5vUP2 LeswtWoBGHGoYXUzAwuxQGRGxoNbh8BROK3gmJ0.

The header and cipher text are concatenated and produce the result shown in the example.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com