

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2017

M. Thomson
Mozilla
P. Beverloo
Google
June 18, 2017

Voluntary Application Server Identification (VAPID) for Web Push draft-ietf-webpush-vapid-03

Abstract

An application server can use the method described to voluntarily identify itself to a push service. This identification information can be used by the push service to attribute requests that are made by the same application server to a single entity. An application server can include additional information that the operator of a push service can use to contact the operator of the application server. This identification information can be used to restrict the use of a push subscription a single application server.

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 December 20, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
1.1.	Voluntary Identification	3
1.2.	Notational Conventions	3
2.	Application Server Self-Identification	4
2.1.	Application Server Contact Information	4
2.2.	Additional Claims	4
2.3.	Cryptographic Agility	5
2.4.	Example	5
3.	Vapid Authentication Scheme	6
3.1.	Token Parameter (t)	6
3.2.	Public Key Parameter (k)	6
4.	Subscription Restriction	7
4.1.	Creating a Restricted Push Subscription	7
4.2.	Using Restricted Subscriptions	8
5.	Security Considerations	9
6.	IANA Considerations	9
6.1.	Vapid Authentication Scheme Registration	9
6.2.	Vapid Authentication Scheme Parameters	10
6.3.	application/webpush-options+json Media Type Registration	10
7.	Acknowledgements	11
8.	References	11
8.1.	Normative References	12
8.2.	Informative References	13
	Authors' Addresses	13

[1.](#) Introduction

The Web Push protocol [[RFC8030](#)] describes how an application server is able to request that a push service deliver a push message to a user agent.

As a consequence of the expected deployment architecture, there is no basis for an application server to be known to a push service prior to requesting delivery of a push message. Requiring that the push service be able to authenticate application servers places an unwanted constraint on the interactions between user agents and application servers, who are the ultimate users of a push service. That constraint would also degrade the privacy-preserving properties the protocol provides. For these reasons, [[RFC8030](#)] does not define a mandatory system for authentication of application servers.

An unfortunate consequence of this design is that a push service is exposed to a greater risk of denial of service attack. While requests from application servers can be indirectly attributed to user agents, this is not always efficient or even sufficient. Providing more information about the application server directly to a push service allows the push service to better distinguish between legitimate and bogus requests.

Additionally, this design also relies on endpoint secrecy as any application server in possession of the endpoint is able to send messages, albeit without payloads. In situations where usage of a subscription can be limited to a single application server, the ability to associate a subscription with the application server could reduce the impact of a data leak.

1.1. Voluntary Identification

This document describes a system whereby an application server can volunteer information about itself to a push service. At a minimum, this provides a stable identity for the application server, though this could also include contact information, such as an email address.

A consistent identity can be used by a push service to establish behavioral expectations for an application server. Significant deviations from an established norm can then be used to trigger exception handling procedures.

Voluntarily-provided contact information can be used to contact an application server operator in the case of exceptional situations.

Experience with push service deployment has shown that software errors or unusual circumstances can cause large increases in push message volume. Contacting the operator of the application server has proven to be valuable.

Even in the absence of usable contact information, an application server that has a well-established reputation might be given preference over an unidentified application server when choosing whether to discard a push message.

1.2. 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](#)].

The terms "push message", "push service", "push subscription", "application server", and "user agent" are used as defined in [\[RFC8030\]](#).

2. Application Server Self-Identification

Application servers that wish to self-identify generate and maintain a signing key pair. This key pair **MUST** be usable with elliptic curve digital signature (ECDSA) over the P-256 curve [\[FIPS186\]](#). Use of this key when sending push messages establishes an identity for the application server that is consistent across multiple messages.

When requesting delivery of a push message, the application includes a JSON Web Token (JWT) [\[RFC7519\]](#), signed using its signing key. The token includes a number of claims as follows:

- o An "aud" (Audience) claim in the token **MUST** include the unicode serialization of the origin ([Section 6.1 of \[RFC6454\]](#)) of the push resource URL. This binds the token to a specific push service. This ensures that the token is reusable for all push resource URLs that share the same origin.
- o An "exp" (Expiry) claim **MUST** be included with the time after which the token expires. This limits the time over which a token is valid. An "exp" claim **MUST NOT** be more than 24 hours from the time of the request.

This JWT is included in an Authorization header field, using an auth-scheme of "vapid". A push service **MAY** reject a request with a 403 (Forbidden) status code [\[RFC7235\]](#) if the JWT signature or its claims are invalid.

The JWT **MUST** use a JSON Web Signature (JWS) [\[RFC7515\]](#). The signature **MUST** use ECDSA on the NIST P-256 curve [\[FIPS186\]](#) which is identified as "ES256" [\[RFC7518\]](#).

2.1. Application Server Contact Information

If the application server wishes to provide contact details it **MAY** include a "sub" (Subject) claim in the JWT. The "sub" claim **SHOULD** include a contact URI for the application server as either a "mailto:" (email) [\[RFC6068\]](#) or an "https:" [\[RFC2818\]](#) URI.

2.2. Additional Claims

An application server **MAY** include additional claims using public or private names (see Sections [4.2](#) and [4.3](#) of [\[RFC7519\]](#)). Since the JWT

is in a header field, the size of additional claims SHOULD be kept as small as possible.

2.3. Cryptographic Agility

The "vapid" authentication scheme is used to identify the specific profile of JWT defined in this document. A different authentication scheme is needed to update the signature algorithm or other parameters. This ensures that existing mechanisms for negotiating authentication scheme can be used rather than defining new parameter negotiation mechanisms.

2.4. Example

An application server requests the delivery of a push message as described in [\[RFC8030\]](#). If the application server wishes to self-identify, it includes an Authorization header field with credentials that use the "vapid" authentication scheme ([Section 3](#)).

```
POST /p/JzLQ3raZJfFBR0aqv0MsLrt54w4rJUsv HTTP/1.1
Host: push.example.net
TTL: 30
Content-Length: 136
Content-Encoding: aes128gcm
Authorization: vapid
  t=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJodHRwczovL3
  B1c2guZXhhbXBsZS5uZXQiLCJleHAiOiJlE0NTM1MjM3NjgsInN1YiI6Im1ha
  Wx0bzpwdXNoQGV4YW1wbGUuY29tIn0.i3CYb7t4xfxCDquptFOepC9GAu_H
  LGkMlMuCGSK2rpiUfnK9ojFwDXb1JrErtmysazNjjvW2L90kSSHvzvoD1oA,
  k=BA1Hxzyi1RUM1b5wjxsn7nGxAszw2u61m164i3MrAIxHF6YK5h4SDYic-dR
  uU_RCPCfA5aq9ojSwk5Y2EmClBPs
  { encrypted push message }
```

Figure 1: Requesting Push Message Delivery with JWT

Note that the example header fields in this document include extra line wrapping to meet formatting constraints.

The "t" parameter of the Authorization header field contains a JWT; the "k" parameter includes the base64url-encoded key that signed that token. The JWT input values and the JWK [\[RFC7517\]](#) corresponding to the signing key are shown in Figure 2 with additional whitespace added for readability purposes. This JWT would be valid until 2016-01-23T04:36:08Z [\[RFC3339\]](#).


```
JWT header = { "typ": "JWT", "alg": "ES256" }
JWT body = { "aud": "https://push.example.net",
             "exp": 1453523768,
             "sub": "mailto:push@example.com" }
JWK = { "crv": "P-256",
        "kty": "EC",
        "x": "DUfHPKLVFQzVvnCPGyfucbECzPDa7rWbXriLcysAjEc",
        "y": "F6YK5h4SDYic-dRuU_RCPcfA5aq9ojSwk5Y2EmClBPs" }
```

Figure 2: Decoded Example Values

3. Vapid Authentication Scheme

A new "vapid" HTTP authentication scheme [[RFC7235](#)] is defined. This authentication scheme carries a signed JWT, as described in [Section 2](#), plus the key that signed that JWT.

This authentication scheme is for origin-server authentication only. Therefore, this authentication scheme MUST NOT be used with the Proxy-Authenticate or Proxy-Authorization header fields.

This authentication scheme does not require a challenge. Clients are able to generate the Authorization header field without any additional information from a server. Therefore, a challenge for this authentication scheme MUST NOT be sent in a WWW-Authenticate header field.

Two parameters are defined for this authentication scheme: "t" and "k". All unknown or unsupported parameters to "vapid" authentication credentials MUST be ignored. The "realm" parameter is ignored for this authentication scheme.

This authentication scheme is intended for use by an application server when using the Web Push protocol [[RFC8030](#)], but it could be used in other contexts if applicable.

3.1. Token Parameter (t)

The "t" parameter of the "vapid" authentication scheme carries a JWT as described in [Section 2](#).

3.2. Public Key Parameter (k)

In order for the push service to be able to validate the JWT, it needs to learn the public key of the application server. A "k" parameter is defined for the "vapid" authentication scheme to carry this information.

The "k" parameter includes an elliptic curve digital signature algorithm (ECDSA) public key [[FIPS186](#)] in uncompressed form [[X9.62](#)] that is encoded using base64url encoding [[RFC7515](#)].

Note: X9.62 encoding is used over JWK [[RFC7517](#)] for two reasons. A JWK does not have a canonical form, so X9.62 encoding makes it easier for the push service to handle comparison of keys from different sources. Secondly, the X9.62 encoding is also considerably smaller.

Some implementations permit the same P-256 key to be used for signing and key exchange. An application server **MUST** select a different private key for the key exchange [[I-D.ietf-webpush-encryption](#)] and signing the authentication token. Though a push service is not obligated to check either parameter for every push message, a push service **SHOULD** reject push messages that have identical values for these parameters with a 400 (Bad Request) status code.

4. Subscription Restriction

The public key of the application server serves as a stable identifier for the server. This key can be used to restrict a push subscription to a specific application server.

Subscription restriction reduces the reliance on endpoint secrecy by requiring proof of possession to be demonstrated by an application server when requesting delivery of a push message. This provides an additional level of protection against leaking of the details of the push subscription.

4.1. Creating a Restricted Push Subscription

The user agent includes the public key of the application server when requesting the creation of a push subscription. This restricts use of the resulting subscription to application servers that are able to provide proof of possession for the corresponding private key.

The public key is then added to the request to create a push subscription. The push subscription request is extended to include a body. The body of the request is a JSON object as described in [[RFC7159](#)]. A "vapid" member is added to this JSON object, containing the public key on the P-256 curve, encoded in the uncompressed form [[X9.62](#)] and base64url encoded [[RFC7515](#)]. The media type of the body is set to "application/webpush-options+json" (see [Section 6.3](#) for registration of this media type).

A push service **MUST** ignore the body of a request that uses a different media type. For the "application/webpush-options+json"

media type, a push service **MUST** ignore any members on this object that it does not understand.

The example in Figure 3 shows a restriction to the key used in Figure 1. Extra whitespace is added to meet formatting constraints.

```
POST /subscribe/ HTTP/1.1
Host: push.example.net
Content-Type: application/webpush-options+json;charset=utf-8
Content-Length: 104

{ "vapid": "BA1Hxzyi1RUM1b5wjxsn7nGxAszw2u61m164i3MrAIxH
          F6YK5h4SDYic-dRuU_RCPcFA5aq9ojSwk5Y2EmClBPs" }
```

Figure 3: Example Subscribe Request

An application might use the Web Push API [[API](#)] to provide the user agent with a public key.

4.2. Using Restricted Subscriptions

When a push subscription has been associated with an application server, the request for push message delivery **MUST** include proof of possession for the associated private key that was used when creating the push subscription.

A push service **MUST** reject a message that omits mandatory credentials with a 401 (Unauthorized) status code. A push service **MAY** reject a message that includes invalid credentials with a 403 (Forbidden) status code. Credentials are invalid if:

- o either the authentication token or public key are not included in the request,
- o the signature on the JWT cannot be successfully verified using the included public key,
- o the current time is later than the time identified in the "exp" (Expiry) claim or more than 24 hours before the expiry time,
- o the origin of the push resource is not included in the "aud" (Audience) claim, or
- o the public key used to sign the JWT doesn't match the one that was included in the creation of the push subscription.

A push service **MUST NOT** forward the JWT or public key to the user agent when delivering the push message.

An application server that needs to replace its signing key needs to create a new subscription that is restricted to the updated key. Application servers need to remember the key that was used when creating a given subscription.

5. Security Considerations

This authentication scheme is vulnerable to replay attacks if an attacker can acquire a valid JWT. Applying narrow limits to the period over which a replayable token can be reused limits the potential value of a stolen token to an attacker and can increase the difficulty of stealing a token.

An application server might offer falsified contact information. A push service operator therefore cannot use the presence of unvalidated contact information as input to any security-critical decision-making process.

Validation of a signature on the JWT requires a non-trivial amount of computation. For something that might be used to identify legitimate requests under denial of service attack conditions, this is not ideal. Application servers are therefore encouraged to reuse tokens, which permits the push service to cache the results of signature validation.

An application server that changes its signing key breaks linkability between push messages that it sends under the different keys. A push service that relies on a consistent identity for application servers might categorize requests made with new keys differently. Gradual migration to a new signing key reduces the chances that requests that use the new key will be categorized as abusive.

6. IANA Considerations

This document registers a new authentication scheme, a registry for parameters of that scheme, and media type for push options.

6.1. Vapid Authentication Scheme Registration

This document registers the "vapid" authentication scheme in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" established in [[RFC7235](#)].

Authentication Scheme Name: vapid

Pointer to specification text: [Section 3](#) of this document

Notes: This scheme is origin-server only and does not define a challenge.

6.2. Vapid Authentication Scheme Parameters

This document creates a "Vapid Authentication Scheme Parameters" registry for parameters to the "vapid" authentication scheme. This registry is under the "WebPush Parameters" grouping. The registry operates on the "Specification Required" policy [[RFC5226](#)].

Registrations MUST include the following information:

Parameter Name: A name for the parameter, which conforms to the "token" grammar [[RFC7230](#)]

Purpose (optional): A brief identifying the purpose of the parameter.

Specification: A link to the specification that defines the format and semantics of the parameter.

This registry initially contains the following entries:

Parameter Name	Purpose	Specification
t	JWT authentication token	[[RFC-to-be]], Section 3.1
k	signing key	[[RFC-to-be]], Section 3.2

6.3. application/webpush-options+json Media Type Registration

This document registers the "application/webpush-options+json" media type in the "Media Types" registry following the process described in [[RFC6838](#)].

Type name: application

Subtype name: webpush-options+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: binary

Security considerations: See [[RFC7159](#)] for security considerations specific to JSON.

Interoperability considerations: See [[RFC7159](#)] for interoperability considerations specific to JSON.

Published specification: This document.

Applications that use this media type: Web browsers, via the Web Push Protocol [[RFC8030](#)].

Fragment identifier considerations: None, see [[RFC7159](#)].

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): .json

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Martin Thomson (martin.thomson@gmail.com)

Intended usage: LIMITED USE

Restrictions on usage: For use with the Web Push Protocol [[RFC8030](#)].

Author: See "Authors' Addresses" section of this document.

Change controller: Internet Engineering Task Force

[7.](#) Acknowledgements

This document would have been much worse than it currently is if not for the contributions of Benjamin Bangert, JR Conlin, Chris Karlof, Costin Manolache, Adam Roach, and others.

[8.](#) References

8.1. Normative References

- [FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.
- [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>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6068] Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", [RFC 6068](#), DOI 10.17487/RFC6068, October 2010, <<http://www.rfc-editor.org/info/rfc6068>>.
- [RFC6454] Barth, A., "The Web Origin Concept", [RFC 6454](#), DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", [BCP 13](#), [RFC 6838](#), DOI 10.17487/RFC6838, January 2013, <<http://www.rfc-editor.org/info/rfc6838>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", [RFC 7159](#), DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [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>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.

- [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>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", [RFC 7518](#), DOI 10.17487/RFC7518, May 2015, <<http://www.rfc-editor.org/info/rfc7518>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", [RFC 7519](#), DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.
- [RFC8030] Thomson, M., Damaggio, E., and B. Raymor, Ed., "Generic Event Delivery Using HTTP Push", [RFC 8030](#), DOI 10.17487/RFC8030, December 2016, <<http://www.rfc-editor.org/info/rfc8030>>.
- [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] Beverloo, P., Thomson, M., van Ouwerkerk, M., Sullivan, B., and E. Fulla, "Push API", May 2017, <<https://w3c.github.io/push-api/>>.
- [I-D.ietf-webpush-encryption]
Thomson, M., "Message Encryption for Web Push", [draft-ietf-webpush-encryption-08](#) (work in progress), February 2017.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", [RFC 7517](#), DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Peter Beverloo
Google

Email: beverloo@google.com