Workgroup:
Grant Negotiation and Authorization Protocol
Internet-Draft:
draft-ozdemir-gnap-spc-extension-00
Published: 13 March 2023
Intended Status: Informational
Expires: 14 September 2023
Authors: O. T. Ozdemir
Fynbos
A. Hope-Bailie
Fynbos

GNAP Secure Payment Confirmation Extension

Abstract

GNAP Secure Payment Confirmation (SPC) Extension is a Grant Negotiation and Authorization Protocol ([GNAP]) extension that defines a method for authentication of the end user during a payment transaction. This extension helps leverage hardware and software authenticators such as biometric scanners while authenticating the end user.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://fynbos-dev.github.io/gnap-spc-extension/draft-ozdemir-gnap-spc-extension.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ozdemir-gnap-spc-extension/.

Source for this draft and an issue tracker can be found at https://github.com/fynbos-dev/gnap-spc-extension.

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 https://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 14 September 2023.

Table of Contents

1. Introduction

GNAP Secure Payment Confirmation Extension is an extension developed on top of the Grant Negotiation and Agreement Protocol [GNAP]. It defines a method for authentication of the end user during a payment transaction using Secure Payment Confirmation ([SPC]). This extension helps leverage authenticators such as fingerprint scanners, facial recognition systems, etc. while authenticating during a GNAP interaction.

Secure Payment Confirmation ([SPC]) is a Web Platform API implemented in Web browsers which allows a website to invoke

[WebAuthn] to both authenticate the end user and confirm the details of the payment during a payment transaction.

A method for detecting this capability in the client software is provided in Appendix A.

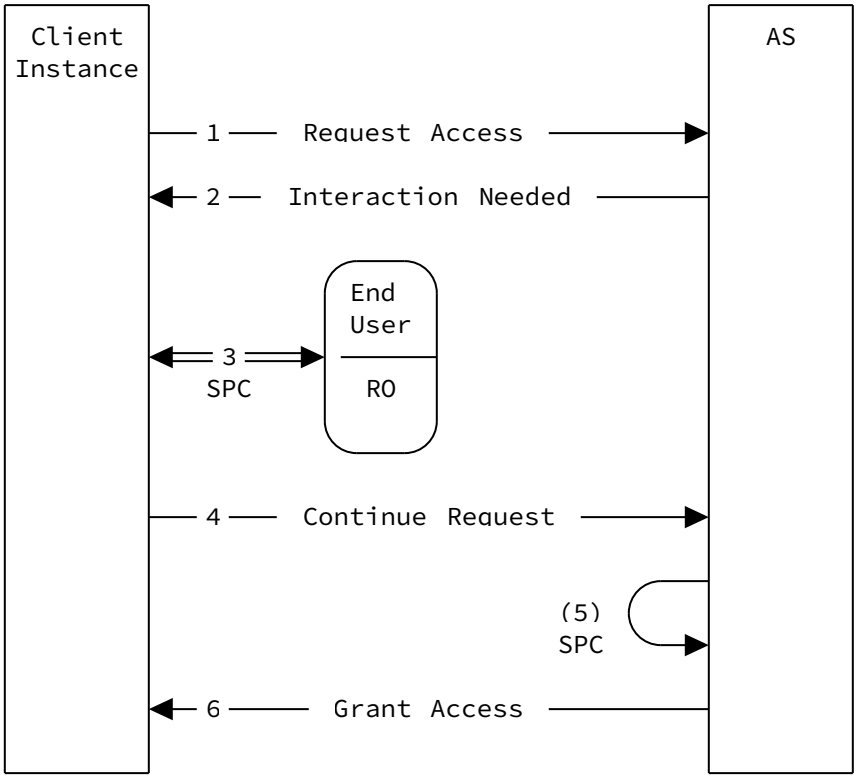
1.1. Conventions and Definitions

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.

2. Secure Payment Confirmation Interaction

When using SPC in [GNAP], the end user is prompted to authenticate during the interaction phase of the protocol, when the grant is in the *pending* state.

The overall flow of the protocol is shown here:



1. The client instance makes a grant request to the AS, indicating that it can support SPC as an interaction start method. The client instance includes an identifier for the intended end

user in the request. The client instance does not include an interaction finish method, since the interaction will happen outside of the AS and the client software will not need to be signaled by the AS to continue the grant request. (Section 2.1)

2. The AS creates a new grant request in the *pending* state and responds to the grant request with a challenge to be signed by a set of candidate credentials that are known to have been provisioned for the end user identified by the client. (Section 2.2)

3. The client instance engages the SPC protocol to sign the challenge. (Section 2.3)

4. The client instance continues the grant request, including the results of the SPC challenge response from (3) in the grant continuation request. (Section 2.4)

5. The AS validates the signature, completing the SPC process and placing the grant in the *approved* state. (Section 3)

6. The AS returns an access token in a standard GNAP grant response.

The end user provides confirmation using their credential, which the client instance presents back to the AS in a continuation response. The AS then verifies this confirmation in order to process the grant request and grant access to the client instance.

2.1. Requesting Credentials

A client instance that is able to use SPC for user interaction can request a grant from an authorization server with SPC as an interaction start method.

The SPC interaction start method is defined as a string type with value spc. If the spc interaction start method is accepted, the corresponding response is returned as discussed in Section 2.2.

When requesting the SPC interaction method the client **MUST** identify the end user using the user property in the grant request object to allow the AS to find the registered credentials for the user. If the user property is not included in the request, the AS **MUST NOT** enable this interaction mode for this request.

A non-normative example of a grant request that uses SPC as its interaction start method is below.

```
"access_token": {
  "access": ["make-payment"]
},
"client": "xyz-client-1234a",
"interact": {
  "start": [
    "spc"
  ]
},
"user": {
  "sub_ids": [{
    "subject_type": "email",
    "email": "user@example.com"
  }]
}
```

2.2. Providing a Credential Challenge

In response to a client instance's grant request, if the AS determines that it has one or more registered SPC credentials of the end user, the AS responds with an spc field in the interact object.

The AS determines the end user using the user property from the grant request. The user property should have the required information such as email addresses, usernames, etc. for determining the end user, see Section 2.4 of [GNAP]. If the user property is not included in the request, it is not possible for the AS to determine the credentials for the end user.

spc (object): An object containing parameters required for performing secure payment confirmation on the end user's device. **REQUIRED** if the AS is allowing SPC interaction for this request.

This object contains the following properties:

credential_ids (array of strings): A list of identifiers of credentials that could potentially be used to respond to this interaction mode. Each credential identifier **MUST** be base64url encoded with no padding. **REQUIRED**.

challenge (string): A random challenge that the relying party (the AS) generates on the server side to prevent replay attacks. The challenge **MUST** be base64url encoded with no padding. **REQUIRED**.

payment_instrument (object): An object describing the payment instrument which will be used to execute the payment. **REQUIRED**.

The payment instrument object has the following properties:

display_name (string):

A display name to use in the SPC user interface. **REQUIRED**.

icon (string): The URL of an icon to display in the SCP user interface. The URL may either identify an image on an internet-accessible server (e.g., https://bank.com/card.png), or directly encode the icon data via a Data URL [rfc2397]. Between the two types of URLs, Data URLs offer several benefits to the Relying Party. They can improve reliability (e.g., in the case that the icon hosting server may be unavailable). They can also enhance validation because the Relying Party has cryptographic evidence of what the browser displayed to the user: the icon URL is signed as part of the authentication ceremony.**REQUIRED**.

icon_must_be_shown (boolean): Indicates whether the specified icon must be successfully fetched and shown for the request to succeed. Defaults to TRUE if not provided. **OPTIONAL**.

A non-normative example of a grant request continue response that uses SPC as its interaction method is below.

```
{
  "interact": {
    "spc": {
      "credential_ids": ["MTIzMjMxMzIyMz..."],
      "challenge": "dGhpcyBpcyBh...",
      "payment_instrument" : {
        "display_name": "Card ending in 4242",
        "icon": "https://wallet.example/card-art.png",
        "icon_must_be_shown": true
      }
    }
  },
  "continue": {
    "access_token": {
      "value": "80UPRY5NM33OMUKMKSKU"
    },
    "uri": "http://wallet.example/continue/5e69f364-b14d-4fdf-8b6b-3b6ff
  }
}
```

2.3. Authenticating User

When the client instance receives an spc interaction response from the AS, the client instance **SHOULD** initiate the authentication ceremony following Section 4 of [SPC].

When performing this ceremony, the client instance decodes the challenge and each credential from credential_ids using base64url and converts them to a buffer for input into the browser API.

The challenge value is used as the challenge input parameter and the credential_ids value is used as the credentialIds input parameter in the SecurePaymentConfirmationRequest dictionary passed to the browser API. The instrument input parameter is constructed by copying the display_name, icon, and icon_must_be_shown properties from the payment_instrument object into a new PaymentCredentialInstrument.

When the client initates the authentication ceremony, the browser API checks if the device has at least one of the credential ids and continues to the authentication ceremony only if the device has one of the credentials. In this phase, the credential that the end user chooses as the authentication method is going to be used for signing the cryptogram.

When the authentication ceremony is complete, the client instance will have access to the response data from the ceremony to be returned to the AS.

2.4. Completing Interaction

Once the authentication ceremony is complete, the client instance continues the grant request by calling the grant continuation URI. The client instance includes a body in the grant continuation request including a field named public_key_cred:

public_key_cred (object): The results of the SPC authentication ceremony in response to the AS, to be used by the AS to authorize the request.

The public_key_cred object contains the following fields as defined by the Web Authentication Assertion object [WebAuthn]:

client_data_json (string): clientDataJSON property from Web Authentication Assertion object. This **MUST** be encoded using base64url. **REQUIRED**.

authenticator_data (string): authenticatorData property from Web Authentication Assertion object. This **MUST** be encoded using base64url. **REQUIRED**.

signature (string): signature property from Web Authentication Assertion object. This **MUST** be encoded using base64url. **REQUIRED**.

user_handle (string): userHandle property from Web Authentication Assertion object. This **MUST** be encoded using base64url. **REQUIRED**.

A non-normative example of an interaction completion response body is below.

```
{
  "public_key_cred": {
    "client_data_json": "ZXhhbXBsZSBjbGllbnRkYXR...",
    "authenticator_data": "YXV0aGVudGljYXRvckRhdGEg...",
    "signature": "c2lnbmF0dXJlIGV4YW...",
    "user_handle": "dXNlckhhbmRsZSBleG..."
  }
}
```

Since the signature is in response to a challenge provided by the AS, the client instance **MUST NOT** send this parameter to a new grant request. The grant request **MUST** be in the *pending* state when this parameter is sent to the AS.

3. Verifying Authentication Assertion

When the AS receives the public_key_cred value in a grant continuation request, the AS **MUST** perform the steps specified in Section 8.1 of [SPC]. Each property of a public key credential returned following successful invocation of the SPC handler client_data_json, authenticator_data, signature and user_handle **MUST** be present as expected for starting verification. The AS **MUST** decode each property of the public key credential in the response using base64url before performing the verification.

The grant request **MUST** be in the *pending* state when this parameter is received in order for it to be processed. If the grant request is in any other state, the AS **MUST** return an error.

The AS **MUST** ensure that the transaction details encoded in the public key credential match the details of the transaction that the client instance is requesting a grant to perform.

If the AS determines that the authorization is sufficient, the AS grants access tokens and releases subject information to the client instance.

4. Security Considerations

TODO Security

5. IANA Considerations

IANA is requested to register the following values into the named registries.

5.1. Interaction Start Modes

IANA is requested to register the following modes into the Interaction Start Modes registry defined by [GNAP].

Mode	Type	Specification document(s)
spc	string	Section 2.1 of RFC nnnn

Table 1

5.2. Interaction Mode Responses

IANA is requested to register the following methods into the Interaction Mode Responses registry defined by [GNAP].

Name	Specification document(s)
spc	Section 2.2 of RFC nnnn

Table 2

5.3. Grant Request Parameters

IANA is requested to register the following parameters into the Grant Request Parameters registry defined by [GNAP].

Name	Type	Specification document(s)
public_key_cred	object	Section 2.4 of RFC nnnn

Table 3

Acknowledgments

TODO acknowledge.

References

Normative References

[GNAP] Richer, J. and F. Imbault, "Grant Negotiation and Authorization Protocol", Work in Progress, Internet-Draft, draft-ietf-gnap-core-protocol-13, 27 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-gnap-core-protocol-13>.

[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/rfc/rfc2119>.

[rfc2397] Masinter, L., "The "data" URL scheme", RFC 2397, DOI 10.17487/RFC2397, August 1998, <https://www.rfc-editor.org/rfc/rfc2397>.

[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/rfc/rfc8174>.

[SPC] "Secure Payment Confirmation", W3C WD secure-payment-confirmation, W3C secure-payment-confirmation, <https://www.w3.org/TR/secure-payment-confirmation/>.

[WebAuthn] "Web Authentication: An API for accessing Public Key Credentials - Level 3", W3C WD webauthn-3, W3C webauthn-3, <https://www.w3.org/TR/webauthn-3/>.

Informative References

[PaymentRequest] "Payment Request API 1.1", W3C WD payment-request-1.1, W3C payment-request-1 1, <https://www.w3.org/TR/payment-request-1.1/>.

Appendix A. Checking Feature Support

This extension only works if the end user's user agent supports the Payment Request API [PaymentRequest] and SPC. To detect whether SPC is supported on the browser, the client instance can send a fake call to canMakePayment().

The following code provides a feature detect function for the Payment Request API and SPC that could be executed on a merchant's website.

```
const isSecurePaymentConfirmationSupported = async () => {
  if (!'PaymentRequest' in window) {
    return [false, 'Payment Request API is not supported'];
  }

  try {
    // The data below is the minimum required to create the request and
    // check if a payment can be made.
    const supportedInstruments = [
      {
        supportedMethods: "secure-payment-confirmation",
        data: {
          // RP's hostname as its ID
          rpId: 'rp.example',
          // A dummy credential ID
          credentialIds: [new Uint8Array(1)],
          // A dummy challenge
          challenge: new Uint8Array(1),
          instrument: {
            // Non-empty display name string
            displayName: ' ',
            // Transparent-black pixel.
            icon: '
          },
          // A dummy merchant origin
          payeeOrigin: 'https://non-existent.example',
        }
      }
    ];

    const details = {
      // Dummy shopping details
      total: {label: 'Total', amount: {currency: 'USD', value: '0'}},
    };

    const request = new PaymentRequest(supportedInstruments, details);
    const canMakePayment = await request.canMakePayment();
    return [canMakePayment, canMakePayment ? '' : 'SPC is not available'
  } catch (error) {
    console.error(error);
    return [false, error.message];
  }
};
```

Authors' Addresses

Omer Talha Ozdemir
Fynbos

Email: omer@fynbos.dev

Adrian Hope-Bailie
Fynbos

Email: adrian@fynbos.dev