Visa Payment Passkey (VPP) Merchant Implementation Guide

Version 1.1

August 2025

Visa Confidential

Important Information on Confidentiality and Copyright

This document is a supplement of the *Visa Core Rules and Visa Product and Service Rules*. In the event of any conflict between any content in this document, any document referenced herein, any exhibit to this document, or any communications concerning this document, and any content in the *Visa Core Rules and Visa Product and Service Rules*, the *Visa Core Rules and Visa Product and Service Rules* shall govern and control.

THIS GUIDE IS PROVIDED ON AN "AS IS," "WHERE IS," BASIS, "WITH ALL FAULTS" KNOWN AND UNKNOWN. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, VISA EXPLICITLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, REGARDING THE LICENSED WORK AND TITLES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.

THE MATERIAL HEREIN MAY CONTAIN DEPICTIONS AND DESCRIPTIONS OF A PRODUCT, SERVICE AND/OR PROGRAM CURRENTLY IN THE PROCESS OF DEVELOPMENT AND DEPLOYMENT, AND SHOULD BE UNDERSTOOD AS A REPRESENTATION OF THE POTENTIAL FEATURES OF THE FULLY-DEPLOYED PRODUCT, SERVICE AND/OR PROGRAM. ANY FINAL VERSION OF SUCH PRODUCT, SERVICE AND/OR PROGRAM MAY NOT CONTAIN ALL OF THE FEATURES DESCRIBED IN THE MATERIAL. WHERE POTENTIAL FUTURE FUNCTIONALITY IS HIGHLIGHTED, VISA DOES NOT PROVIDE ANY WARRANTY ON WHETHER SUCH FUNCTIONALITY WILL BE AVAILABLE OR IF IT WILL BE DELIVERED IN ANY PARTICULAR MANNER OR MARKET. PARTICIPATION IN PRODUCTS, SERVICES AND/OR PROGRAMS, IF AND WHEN OFFERED, MAY BE SUBJECT TO VISA'S TERMS AND CONDITIONS.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN: THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. VISA MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Note: This document is not part of the Visa Rules. In the event of any conflict between any content in this document, any document referenced herein, any exhibit to this document, or any communications concerning this document, and any content in the Visa Rules, the Visa Rules shall govern and control.

If you have technical questions or questions regarding a Visa service or capability, contact your Visa representative.

Contents

1. Visa Payment Passkey Merchant Implementation Guide

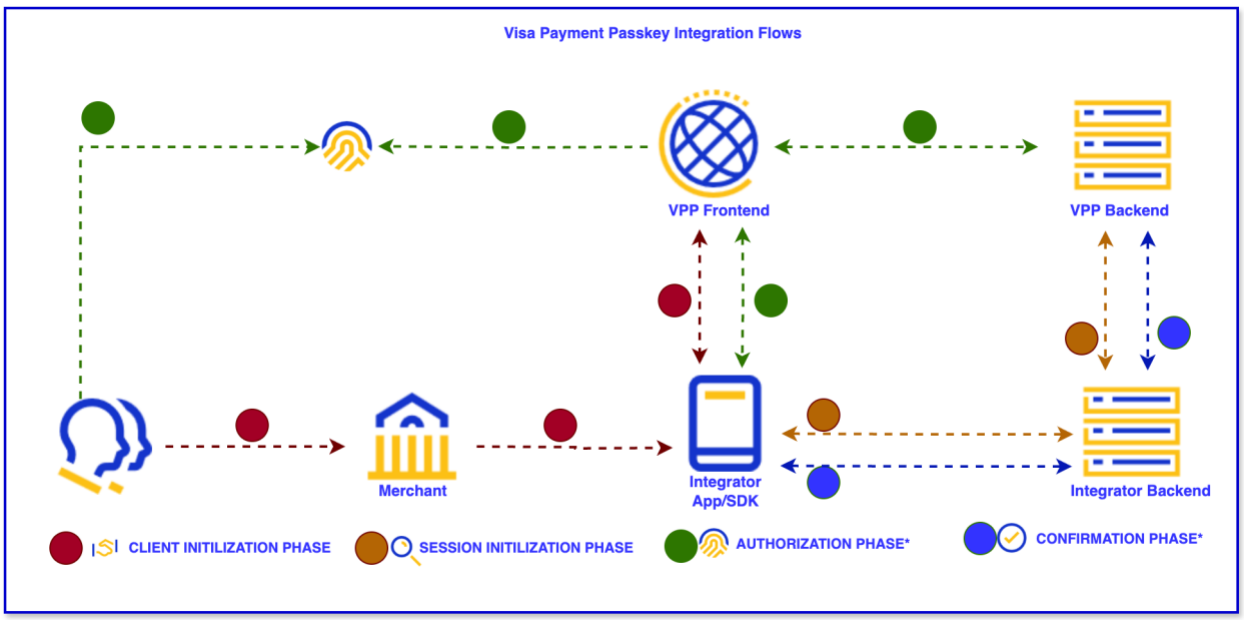
This chapter provides an overview of Visa Payment Passkey (VPP) and includes:

Introduction—Describes VPP purpose, functionality, and benefits.

1.1 Introduction

VPP enables seamless authentication of cardholders across all payment channels, including (but not limited to) guest checkout and card-on-file. A payment passkey represents a strong binding of a device to a Visa payment credential, introducing cardholder authentication without adding unnecessary friction. Built using the FIDO (Fast Identity Online) standard and leveraging Visa's device intelligence, VPP increases confidence in payment authentication. Using the device unlock method (facial scan, fingerprint, pattern, etc.) the cardholder authenticates locally on the device (biometric data is never collected) and cryptographic proof of the authentication is provided to the issuer upon completion.

The following document outlines the functionality available to payment facilitators who perform payments on behalf of merchants, to create new payment passkeys for issued cards through Visa and use registered payment passkeys to perform authentication in the event of a step-up. This is specific to the payment authentication processing and not related to the payment passkey lifecycle management.



2 Integration Steps

This section contains the steps necessary for integrating with VPP.

2.1 Prerequisites

To ensure a seamless integration with VPP, the following prerequisites must be met:

- The user's device must have a platform authenticator such as biometrics including face recognition, fingerprint recognition, and/or screen lock enabled. Currently, VPP does not provide support to roaming authenticators such as YubiKey.
- The browser being used must have WebAuthn API support.

To access the VPP APIs, you will need to create an account on Visa Developer Center (VDC), request access to the Visa Payment Passkey Product, and create a project on VDC. Refer to https://developer.visa.com/pages/working-with-visa-apis/visa-developer-quick-start-guide for more details on getting started.

2.2 Initialization

All merchants must initiate the cardholder session by performing the Initialization step, which verifies support for the required prerequisites. During this step, three key actions are performed:

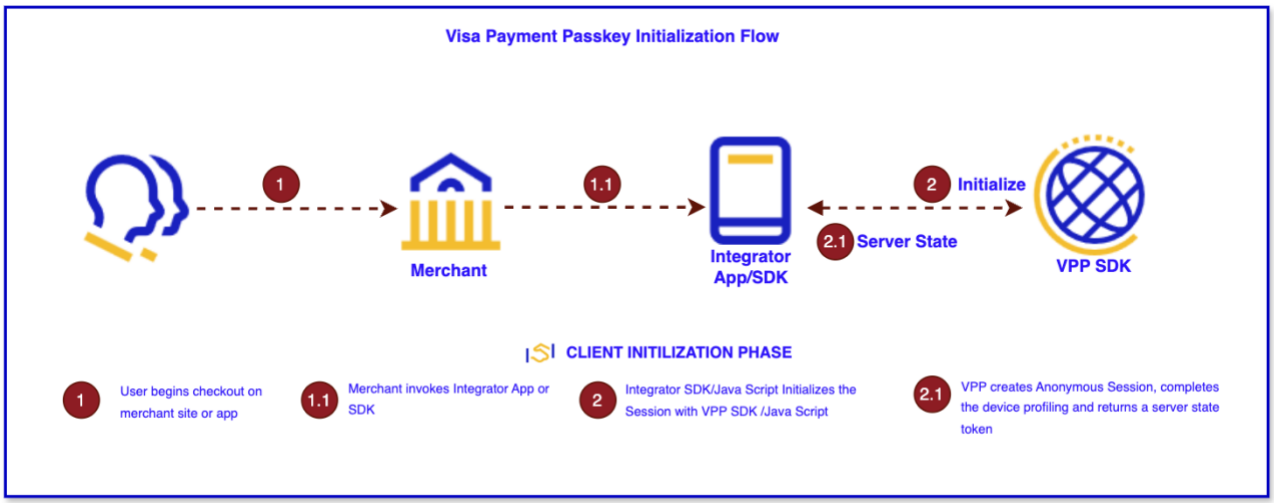
1. FIDO Eligibility Check – Determines whether the device supports FIDO authentication.
2. Device Profiling – Collects various attributes from the device to generate a dfp_session_id, a unique ID representing the device profiling session that is used throughout the rest of the VPP workflow.
3. Anonymous Session Creation – An anonymous session is created and the dfp_session_id and other integrator details are assigned to this session.

These three actions will return a total of two separate results via postMessage events. Examples of both can be found in Section 2.6 Client Side Results.

- The first response is the Initialization Result, which indicates whether the device is FIDO eligible. If the initialization is successful, you will receive a server_state token along with guidance to load VPP in either an iframe or popup during the Authorization_Request step.

- The second response relates to the status of the Device Profiling attempt. If profiling is successful, you will receive a DEVICE_DATA_CAPTURED event. If there are any issues with profiling, you will receive a DEVICE_DATA_CAPTURE_FAILED event.

NOTE: A raw DeviceID is not returned in the device profiling event. Instead, the relevant information including a dfp_session_id is embedded in a server state token and returned as part of the client-side response to a successful Initialization Command. This server state token must be included in subsequent server-side API calls. To successfully proceed to the next step in the workflow, you need both the Initialization and Device Profiling to be successful.



NOTE: It is only expected for the Initialization step to be performed once during a single session regardless of the number of actions requested (timeout = 900 seconds). If the timeout limit is reached, you will need to initialize the cardholder session once again.

```
<iframe src="client.com" />

VPP iframe in hidden mode:  <iframe src=" https://auth.visa.com/oauth2/authorization/request/hub#msg=<uri-fragment>"

** When client sends INITIALIZATION command, the VPP iframe will be in hidden mode. When client sends the AUTHORIZATION_REQUEST
then the client must move the VPP iframe to visible by setting the relevant CSS properties, including setting the appropriate height and width. **
```

To complete the required implementation for Initialization, follow these steps:

- Client creates a hidden iframe (visibility=hidden, height=0, width=0) with src set to `<base_url>/oauth2/authorization/request/hub#msg=<initialization-uri-fragment>`
- The client sends the Initalization command (Step 2 of Initialization Flow) to VPP using the URI fragment. This value must be stringified using JSON.stringify() and then URL encoded using encodeURIcomponent.

Sample Initialization Command:

The placeholders in the example command will need to be updated with the correct values provided during onboarding.

```
<!Doctype html>
<html>
  <head>
    <script>
        const VPP_BASE_URL= `https://sandbox.auth.visa.com`;
        const VPP_INIT_URL =
`${VPP_BASE_URL}/oauth2/authorization/request/hub#msg=`;
        const INIT_COMMAND = {
            "type": "COMMAND",
            "ref": "38aa1649-100b-46d5-a0ec-2d38a34bb5c7", //this ref is used as
correlation id for the API calls made during INITIALIZATION flow.
            "ts": 1752835067477, //current time in millis
            "command": {
                "type": "INITIALIZATION",
                "data": {
                        "response_mode": "com_visa_web_message",
                        "redirect_uri": "url_where_you
want_response_back_in_redirection_flow",
            "session_context": {
                "apn": "APN_PROVIDED_DURING_ONBOARDING_IN_LOWER_CASE",
                "client_software": {
                    "top_origin": "top_origin_or merchant_origin",
                    "integrator_origin":
"fully_qualified_url_where_you_want_post_message_response",
                    "uebas": [
```

```
                        {
                            "source": "VDI",
                            "ref": "DFP_SESSION_ID"
                        }
                    ],
                    "id": "ID_PROVIDED_DURING_ONBOARDING",
                    "version": "VERSION_PROVIDED_DURING_ONBOARDING",
                    "oauth2_version": "1.0",
                    "tenancy": {
                        "product_code": "PRODUCT_CODE_PROVIDED_DURING_ONBOARDING"
                    }
                }
            },
            "response_type": "urn:ext:oauth:response-type:server_state"
        }
    }
}

        window.addEventListener("message", (event) => {
            if (event.data.type === "RESULT") {
                if (event.data.result.command_type === "INITIALIZATION") {
                    if (event.data.result.status === "SUCCESS") {
                        // Initialization Success
                        // Check for iframe_support.binding and
iframe_support.authentication, if true the respective flows can be launched as an
iframe else it has to be pop-up.
                    } else {
                        // Initialization Failure
                    }
                }
            } else if (event.data.type === "EVENT") {
                if (event.data.event.type === "DEVICE_DATA_CAPTURED") {
                    // Device Profiling is success
                } else if (event.data.event.type === "DEVICE_DATA_CAPTURE_FAILED")
{
                    // Device Profiling Failed
                }
            }
```

```
        });

  const fragment = encodeURIComponent(JSON.stringify(INIT_COMMAND));
        const url = `${VPP_INIT_URL}${fragment}`;
        document.getElementById("vpp-iframe").src = url;
    </script>
  </head>
  <body>
      <iframe id="vpp-iframe" src="" style="visibility:hidden;" height=0
width=0></iframe>
  </body>
</html>
```

Following the completion of the client-side Initialization scripting, the client needs to make the Authentication using Passkey API call.

2.3 Iframe Permission Requirements

If you intend to embed the authentication or registration flow in an iframe, you must grant the iframe—and every ancestor frame—the permissions needed for VPP to complete its initialization and authorization requests.

- The integrator's iframe (and every parent iframe) must include: allow="payment *; publickey-credentials-get *; publickey-credentials-create *" referrerpolicy="strict-origin-when-cross-origin"

- If you declare the iframe with a sandbox attribute, add these flags so VPP has the required access: sandbox="allow-scripts allow-same-origin allow-forms allow-popups allow-popups-to-escape-sandbox allow-top-navigation allow-top-navigation-by-user-activation allow-payment allow-modals"

2.4 API Authentication

VPP APIs currently offer the following type of authentication:

- All APIs require Two-Way SSL or X-Pay Token authentication. You can access the Credentials from the sidebar of your Visa Developer Center (VDC) project dashboard. Refer to https://developer.visa.com/pages/working-with-visa-apis/visa-developer-quick-start-guide#section4 for more details.
- Visa Payment Passkey also requires Message Level Encryption for enhanced security for message payloads by using an asymmetric encryption technique (public-key cryptography). Refer to https://developer.visa.com/pages/encryption_guide for more details.

2.5 API Endpoints

The following endpoints are used across different VPP Use Cases. They will be referenced by **API Description** and full details can be found in the *API Reference* document.

API Description*	Endpoint	Verb
Create Passkey	https://api.visa.com/vpp/v1/passkeys/oauth2/authorization/request/pushed	POST
Authentication using Passkey	https://api.visa.com/vpp/v1/passkeys/oauth2/authorization/request/pushed	POST

NOTE: The same endpoint is used for both Registration and Authentication use cases but the request payloads will differ based on the specific scenario.

The VPP base_url is environment specific and is defined as follows:

- Sandbox (Global) – https://sandbox.auth.visa.com
- Sandbox (India) – https://sandbox.in.auth.visa.com
- Production (Global) – https://auth.visa.com
- Production (India) – https://in.auth.visa.com

2.6 Client-Side Results

2.6.1 Initialization Result Data

(Step 2.1 of Initialization Flow)

```
{
  "type": "RESULT",
  "ref": "7e3f652f-e78b-49a5-b8e1-b764aa7ec078",
  "ts": <timestamp_in_millis>,
  "result": {
    "command_type": "INITIALIZATION",
    "command_ref": "38aa1649-100b-46d5-a0ec-2d38a34bb5c7", //This ties the result
against the command that was sent as an uri fragment.
    "status": "SUCCESS", //if status is failure either device eligibility failed
or some internal server error occurred. FIDO flow should be terminated.
    "data": {
      "iframe_support": {
        "authentication": false, //if true then authentication flow can run in an
iframe.
        "binding": false, //if true then binding flow can run in an iframe.
        "options": { //this options is applicable when the iframe support for
respective flows are true.
          "display_mode": {
            "binding": [
              "visible" //default, the binding iframe must be visible
            ],
            "authentication": [
              "visible", //default, the authentication iframe must be visible
              "hidden" //the authentication iframe can be hidden, if the array
includes hidden value.
            ]
          }
        }
      },
      "tokens": [
        {
```

```
          "token": "server_state_token", //this is to be passed to the backend for
the Pushed Authorization request call.
          "token_type": "Bearer", //token type
          "expires_in": 480, // token validity
          "token_type_hint": "urn:ext:oauth:token-type-hint:server_state" //this
tells whether the token generated is an access token or server_state or idtoken
        }
      ],
      "x_via_hint": "1_k_a7mA", //this must be passed to the backend along with
server_state, if received for data center affinity.
      "screen_size": { //this is applicable for both iframe and popup. This
defines the preferred size for the respective flows.
        "binding": {
          "height": 600, //recommended height for binding screen
          "width": 500 //recommended width for binding screen
        },
        "authentication": {
          "height": 480, //recommended height for authentication screen
          "width": 400 //recommended width for authentication screen
        }
      }
    }
}
```

200 The result data for a successful INITIALIZATION command.

Field	Type	Description
iframe_support	object	Indicates whether iframe is supported for different operations
iframe_support.binding	boolean	Whether iframe is supported for binding operations
iframe_support.authentication	boolean	Whether iframe is supported for authentication operations
iframe_support.options	object	Additional options for iframe support
iframe_support.options.display_mode	object	Display mode options for binding and authentication. For binding display will always be visible but for Authentication depending on eligibility the display can contain hidden as well meaning the integrator product can choose to launch authentication iframe as hidden iframe. This will be SPC case.
is_iframe_supported	boolean	Overall indication if iframe is supported. To be Deprecated.
tokens	array	List of tokens returned by the initialization
tokens[].token	string	The actual token value
tokens[].token_type	string	The type of the token (e.g., "Bearer")
tokens[].expires_in	number	Token expiration time in seconds
tokens[].token_type_hint	string	Hint about the type of token
x_via_hint	string	Hint for data center affinity. Only if token is present
screen_size	object	Recommended screen sizes for binding and authentication
screen_size.binding	object	Recommended screen size for binding operations
screen_size.authentication	object	Recommended screen size for authentication operations

2.6.2 Authorization Result Data

2.6.2.1 Authorization Result via postMessage response

(Step 4 of Passkey Create Flow) – iframe based integration with onboarding screen example

```
{
    "type": "RESULT", // "RESULT"
        "ref": crypto.randomUUID, // "<uuid>"
        "ts": Date.now(), // When this result was generated
        "result": {
            "command_type": "AUTHORIZATION_REQUEST", // This Result is against
which command.
            "command_ref": "This result is against which command reference ID.",
            "status": "SUCCESS", // "SUCCESS" or "FAILURE"
            "data": {
                "tokens": [
                    {
                        "token": string, //this must be passed to the backend and
is one time use only.
                        "token_type": TokenType,
                        "expires_in": number,
                        "token_type_hint": TokenHintType,
                        "token_data": {
                            "state": string
                        }
                    }
                ],
                "x_via_hint": string //this must be passed to the backend for data
center affinity
        }
    }
}
// Note:
// 1. Replace `crypto.randomUUID` with a valid UUID generation method in your
environment.
// 2. Replace `string`, `TokenType`, and `TokenHintType` with actual values or
types as per your application requirements.
```

```
// 3. Ensure that the `Date.now()` is replaced with a proper timestamp format if
required.
// 4. The `ref` field should be a unique identifier for the result, typically a
UUID.
// 5. The `command_ref` should match the reference ID of the command this result
is associated with.
// 6. The `status` field indicates whether the authorization request was
successful or failed.
// 7. The `expires_in` field indicates the lifetime of the token in seconds.
// 8. The `x_via_hint` is used for data center affinity and should be set passed
in the B2B calls.
```

2.6.2.2 Authorization Result via form_post response

```
<!Doctype html>
<html>
  <body>
    <form ref={formRef} id='b2cFormPost'
action={redirect_uri_passed_in_par_payload} method='post' className='b2c-form-
post'>
      <input type='hidden' id='code' name='code' value={code} />
      <input type='hidden' id='state' name='state' value={state} />
      <input type='hidden' id='x_via_hint' name='x_via_hint' value={xViaHint} />
    </form>
  </body>
</html>
```

2.6.3 Error Result Data

2.6.3.1 Error via postMessage response

```
{
    "type": "RESULT", // "RESULT"
    "ref": crypto.randomUUID, // "<uuid>"
    "ts": Date.now(), // When this result was generated
    "result": {
        "command_type": "AUTHORIZATION_REQUEST", // This Result is against which
command.
        "command_ref": "This result is against which command reference ID.",
```

```
        "status": "FAILURE",
        "data": {
            "error": "400",
            "error_description": "BAD_REQUEST",
            "error_details": {
                "error": "INVALID_REQUEST",
                "error_description": "Invalid request parameters or format"
            }
        }
    }
}
```

2.6.3.2 Error via form_post response

```
<!Doctype html>
<html>
  <body>
    <form ref={formRef} id='b2cFormPost'
action={redirect_uri_passed_in_par_payload} method='post' className='b2c-form-
post'>
      <input type='hidden' id='error' name='error' value='BAD_REQUEST' />
      <input type='hidden' id='error_description' name='error_description'
value='INVALID_REQUEST' />
      <input type='hidden' id='error_details' name='error_details'
value='USER_ABORTED' />
    </form>
  </body>
</html>
```

For more details on Error Codes, refer to section 4 Error and Event Details

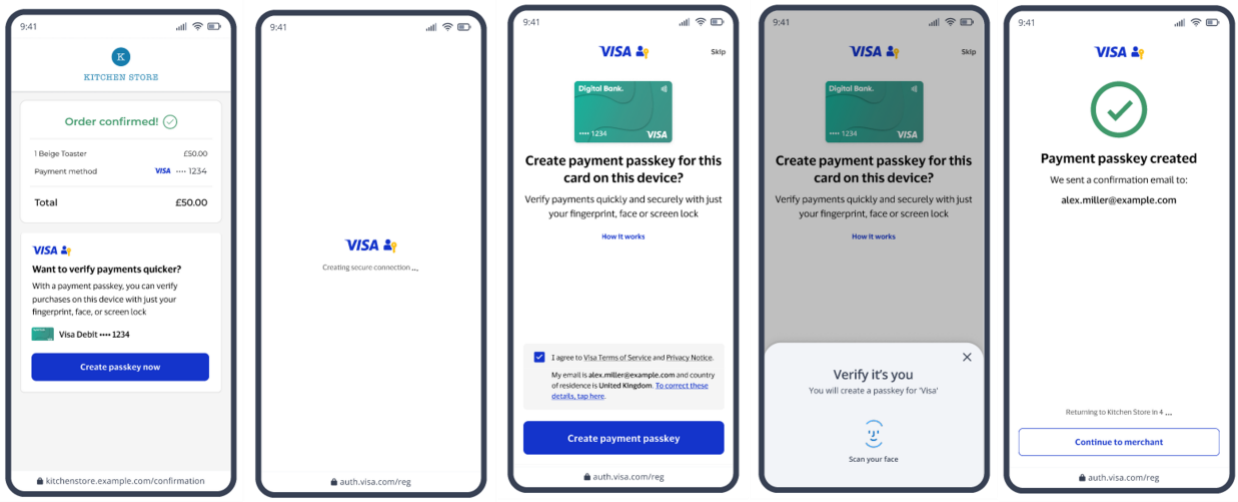
2.6.4 Device Data Event

```
{
  "type": "EVENT",
  "ref": "a7c2b776-ddcd-482b-849d-881da5d38be4", //guid
  "ts": 1752764493094, //timestamp in millis when this event was generated
  "event": {
```

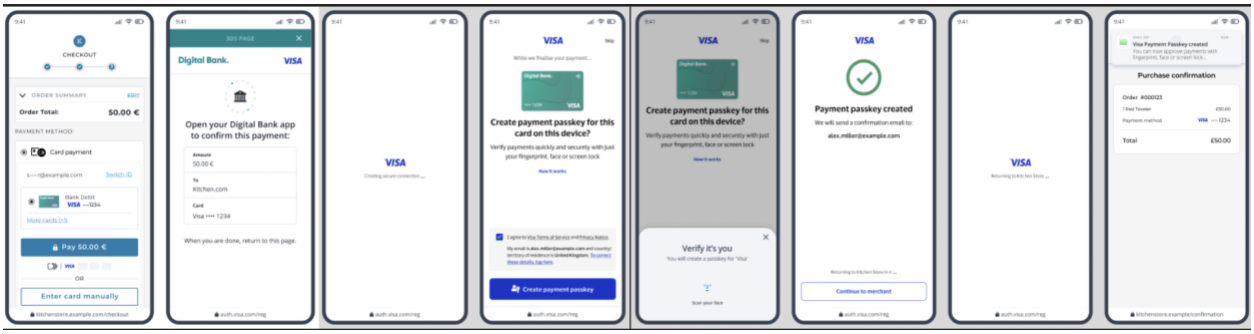
```
      "type": "DEVICE_DATA_CAPTURED", //for success
      "data": {
        "uebas": [
          {
            "ueba_ref": "229cb6d4-7528-4ca7-9fe7-15d3c81d084a", //DFP_SESSION_ID
            "ueba_hint": "FLOW_TYPE",
            "ueba_source": "VDI"
          }
        ],
        "status": "DEVICE_DATA_CAPTURED"
      }
    }
}
// This is a sample event for device data captured
// The event type is DEVICE_DATA_CAPTURED
// ueba_ref is your dfp_session_id
```

3 Use Cases

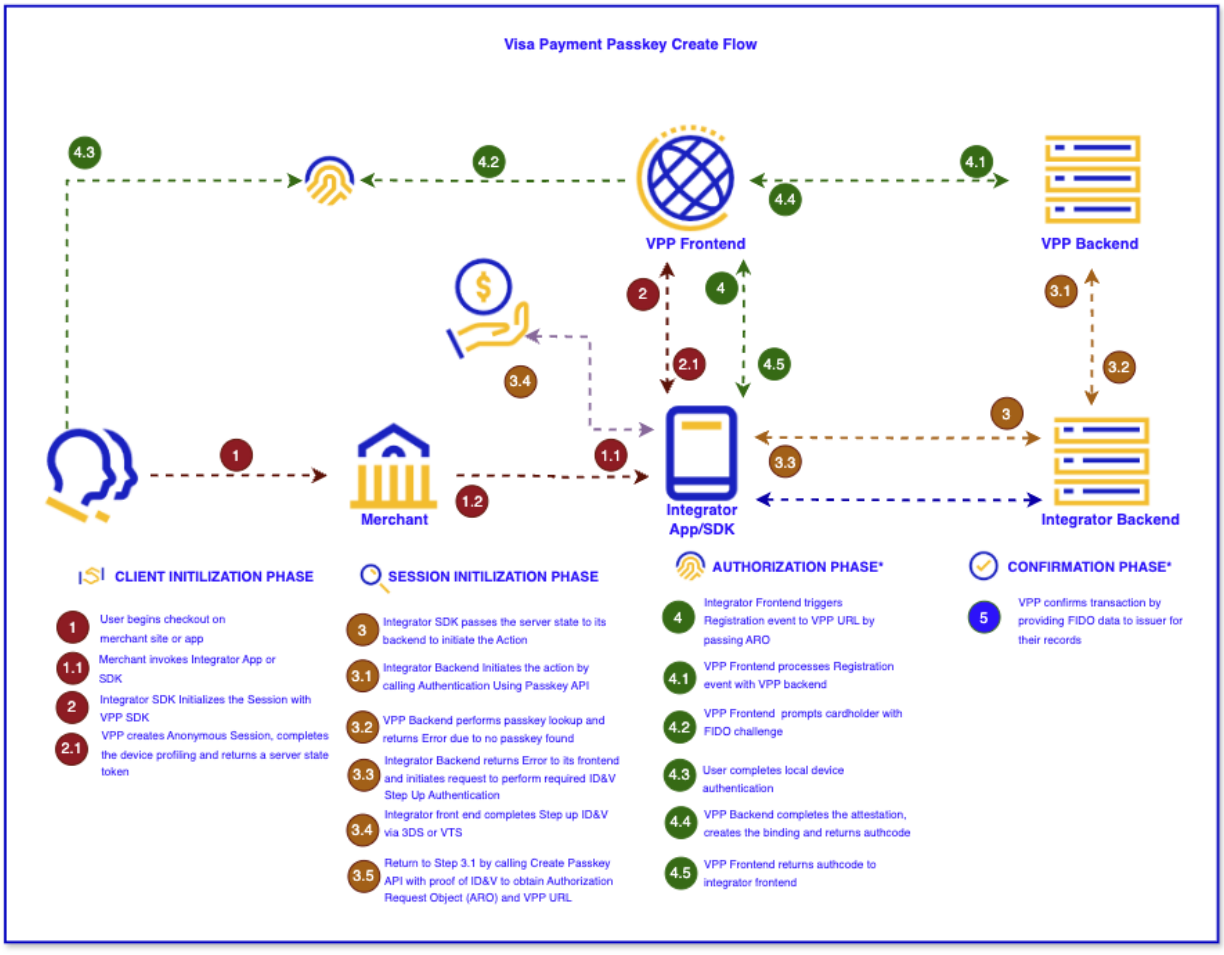
3.1 Registration/Create Passkey



Post payment flow illustration in which cardholder is prompted for passkey creation after transaction



Mid payment flow illustration in which cardholder is prompted for passkey in the same window as the step-up authentication before the transaction is complete



1. The cardholder begins checkout on the merchant site or application (Step 1 of Passkey Create Flow).
2. The integrator triggers the VPP Initialization step (Step 2 of Passkey Create Flow).
3. The integrator calls the Authentication Using Passkey API with payment details to determine if a passkey exists. The required payload for authentication containing the server_state_token from the Initialization result should be included with the type set to com_visa_payment_credential_authentication, the response_mode set to 'form_post' and a valid redirect_uri provided. The response of this request will determine if the cardholder should be directed to either the registration or authentication flow. (Step 3 - 3.2 of Passkey Create Flow).
4. If a 400 failure response is returned, and includes the error "notfound_amr_values", then it signals that a passkey does not exist for the given payment credential and device combination and the cardholder must successfully complete issuer ID&V (i.e. 3DS challenge) before returning to the Visa Payment Passkey registration. For any other 400 response, the flow should terminate here, and the user should be presented with the appropriate fallback option as defined by the merchant and/or integrator. (Step 3.3 - 3.4 of Passkey Create Flow).
5. After a successful issuer ID&V (i.e. 3DS challenge), the integrator should trigger a new request to the Create Passkey API with the required trustchain, email address and proof of ID&V (Step 3.5 of Passkey Create Flow).
6. VPP services use the data collected to check the database for matching records to determine the response to the Create Passkey API request.
7. VPP services responds with:
    - *200*: Is eligible for FIDO Registration and provides a Token and url path to perform registration (Step 3.5 of Passkey Create Flow).
    - Error: If not eligible or request was not formatted correctly.
8. Depending on the type of integration, the merchant should load the VPP URL on their site or app. (Step 4 – 4.1 of Passkey Create Flow).
    - For a post payment flow in which the cardholder is prompted to create a VPP after a completed transaction, the integrator should load VPP as an iframe.

        - The iframe will be a visible iframe.
        - The height of the iframe will be 269px and width will be 343px.
        - The iframe src will be `<base_url>/oauth2/authorization/request/hub#msg=<authorization-request-uri-fragment>`
        - In the Authroization_Request, the response_mode should be set to "com_visa_web_message."

    - For mid payment flow in which the cardholder is prompted to create a VPP before the transaction is complete, the integrator should redirect to VPP in the same ID&V popup window or launch a new popup.

- For the best experience, the recommended height of the popup window is 600px and width is 500px.
- The popup url will be `<base_url><authorization-endpoint-from-par>#msg=<authorization-request-uri-fragment>`
- In the Authroization_Request, the response_mode should be set to form_post.

9. The cardholder is prompted for VPP enrollment and redirected to VPP's UI. On the registration screen, the cardholder is asked to provide consent to enroll into VPP and to consent to a binding of their Payment Credential & Device (Step 4.2 of Passkey Create Flow).
10. VPP, as a Relying Party, sends to the user's browser a challenge and relying party info, where the cardholder performs device authentication (with their Biometric/PIN, but this never leaves the device*) (Step 4.2 - 4.3 of Passkey Create Flow).
11. Upon successful authentication, a new private-public keypair** and attestation is created***. The new public key, credential ID and attestation is sent back to VPP, and a binding is created (Step 4.4 of Passkey Create Flow).
12. A success message is returned based on the response_mode and integration type (Step 4.4 - 4.5 of Passkey Create Flow).

Sample Authorization Request to load VPP prompt:

```
<!Doctype html>
    <html>
        <head>
            <script>
                const VPP_BASE_URL= `https://auth.visa.com`.
                const AUTHORIZATION_REQUEST_COMMAND = {
                    "type": "COMMAND",
                    "ref": crypto.randomUUID(),
                    "ts": Date.now(),
                    "command": {
                        "type": "AUTHORIZATION_REQUEST",
                        "data": {
                            "request":
"<authorization_request_object_in_response_from_par_api>", //This is the ARO
                            "authorization_endpoint":
"/oauth2/authorization/request/hub/payment_credential_binding"
                        }
```

```
                    }
                }
                const fragment =
encodeURIComponent(JSON.stringify(AUTHORIZATION_REQUEST_COMMAND));
                const VPP_AR_URL =
`${VPP_BASE_URL}${AUTHORIZATION_REQUEST_COMMAND.command.data.authorization_request
}#msg=${fragment}`;
                const createPasskey = () => {
                    const top1 = 120;
                    const window_name ='passkeyenrolment';
                    const width = 500;
                    const height = 600;
                    const left1 = (window.screen.width - width) / 2;
                    const window_settings =
`width=${width},height=${height},top=${top1},left=${left1},scrollbars=yes,resizabl
e=no,status=yes,location=no`;
                    window.open(`${VPP_AR_URL}`, window_name, window_settings);
                }
            </script>
        </head>
        <body>
            <button onClick=createPasskey()>Create Passkey</button>
        </body>
  </html>
```

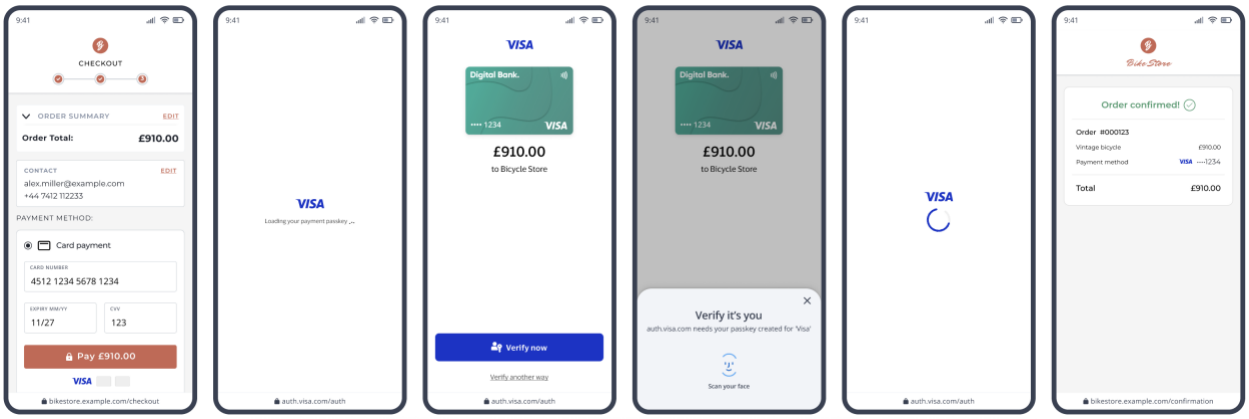
After successful passkey creation, Visa Payment Passkey will send a copy of the attestation including the public key to the issuer of the payment credential.

* Visa will never access, receive, or store biometric information. This is always held on the device, stored securely in the secure element of the device in accordance with the device's operating system.

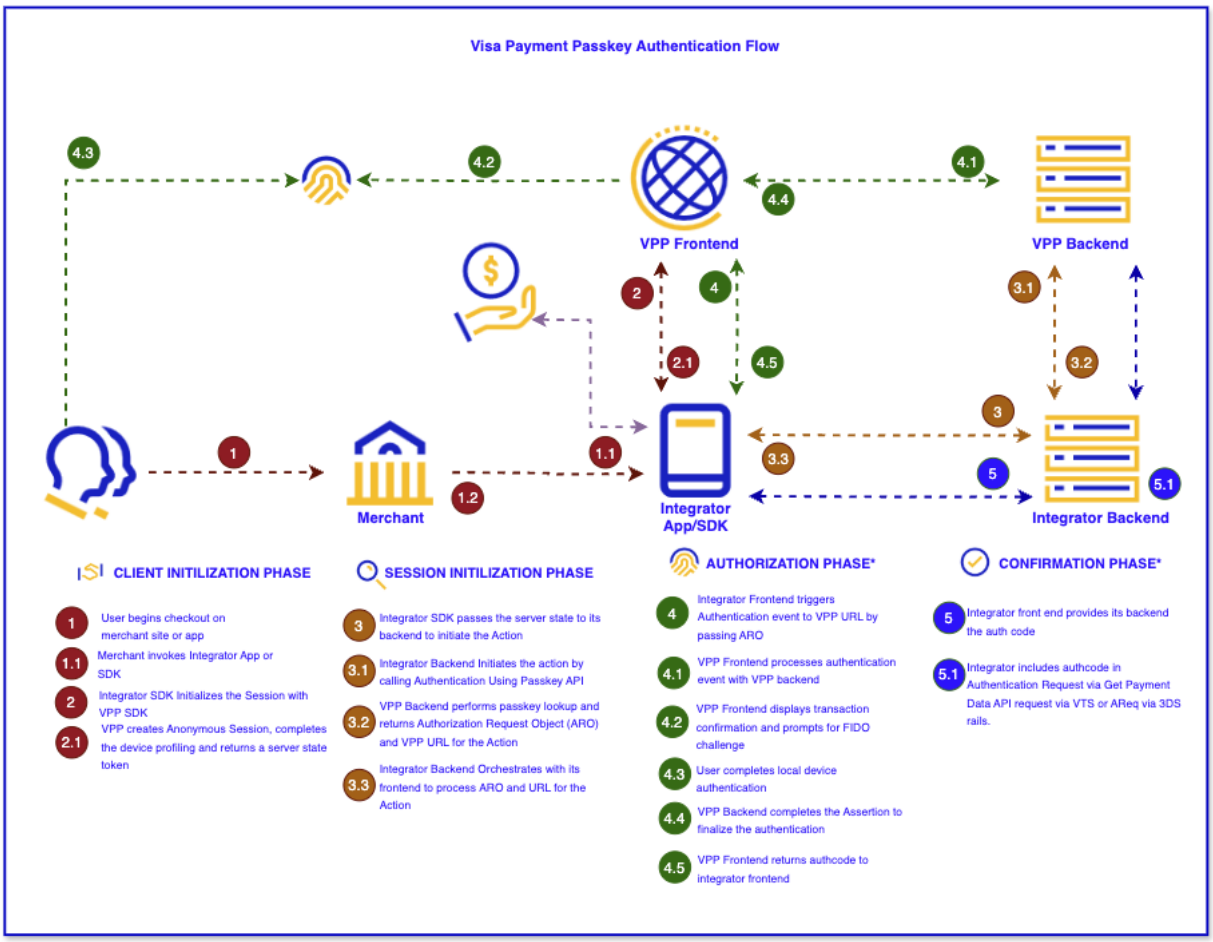
** Private Key: This is a cryptographic key that sits on the device or user's cloud account which is generated when the cardholder registers for FIDO. This allows for transactions made using Biometrics to be authenticated as it generates a signature that can be matched to the Public Key. Public Key: This is a signature verification tool which is meaningless without further information. In itself it cannot identify an individual but is stored with other personal information. The Public key is used to validate the signature generated by the Private Key.

*** In case of a synced passkey, a new Public key pair and Attestation will not be created. Instead, VPP will link the prior passkey from the other device to this device to be used.

3.2 Passkey Authentication Flow



A cardholder authenticates by being redirected to VPP



1. The cardholder begins checkout on merchant site or application (Step 1 of Passkey Create Flow).
2. The integrator triggers the VPP Initialization step (Step 2 of Passkey Authentication Flow).
3. The integrator calls the Authentication Using Passkey API with payment details to determine if a passkey exists. The required payload for authentication containing the server_state_token from the Initialization result should be included with the type set to com_visa_payment_credential_authentication, the response_mode set to 'form_post' and a valid redirect_uri provided. The response of this request will determine if the cardholder should be directed to either the registration or authentication flow. (Step 3 - 3.2 of Passkey Create Flow).
4. VPP services responds with:
    a. 200: Is eligible for FIDO Authentication and provides an ARO and authorization_endpoint to perform authentication. (Step 3.2 of Passkey Create Flow)
    b. 400:

   i. If error includes “notfound_amr_values” the integrator should proceed with the passkey creation flow after successful issuer ID&V (revert back to Section 3.1 for passkey creation flow).
   ii. For any other 400 response, the flow should terminate here, and the user should be presented with the appropriate fallback option as defined by the merchant and/or integrator.
5. Once the cardholder selects the checkout button, the merchant or integrator must launch a popup to prevent the popup from getting blocked.
6. The merchant loads the VPP URL on their site or app. The cardholder is prompted for VPP authentication and redirected to VPP's UI (Step 4 - 4.2 of Passkey Authentication Flow).
   a. The integrator should redirect to VPP using fully formed VPP URL.
   b. Recommended height is 480px and width is 400px for the popup window for better user experience.
   c. The popup url will be `<base_url><authorization-endpoint-from-par>#msg=<authorization-request-uri-fragment>`
   d. In the Pushed Authorization request set response_mode as “form_post” for this flow in the payload.
7. VPP, as a Relying Party, sends to the user’s browser a challenge and relying party info, where the cardholder performs device authentication (with their Biometric/PIN, but this never leaves the device*) (Step 4.1 - 4.3 of Passkey Authentication Flow).
8. Upon successful authentication, a success message is returned (Step 4.4 – 4.5 of Passkey Authentication Flow) along with an auth code.
9. The integrator should include the auth code in their payment authentication request to VTS via the Get Payment Details API or their AReq via the 3DS Authentication Rails (Step 5 of Passkey Authentication Flow).

* Visa will never access, receive, or store biometric information. This is always held on the device, stored securely in the secure element of the device in accordance with the device’s operating system.

4 Error and Event Details

4.1 Event Types

S. No	Event Type	Description
1	DEVICE_DATA_CAPTURED	The integrator SDK receives an EVENT from VPP SDK as a postMessage when the device profiling is captured.
2	DEVICE_DATA_CAPTURE_FAILED	The integrator SDK receives an EVENT from VPP SDK as a postMessage when the device profiling has failed.
3	POPUP_WINDOW_TERMINATED	The integrator SDK receives an EVENT from VPP SDK as a postMessage when the Popup window is terminated.

4.2 Message Types

S. No	Message Type	Description
1	EVENT	These are triggered from VPP Iframe or Popup to parent or window opener based on the context from where they are triggered.
2	COMMAND	These are sent by the integrator to VPP as URI fragment and contains info regarding operation VPP must do
3	RESULT	For each command VPP returns a result with success or failure status and a token for the B2B call.

4.3 Error Types

S. No	Error Types	Description
1	USER_ABORTED	User has aborted the flow.

2	WEBAUTHN_NOT_SUPPORTED	Web AuthN is not supported on this device.
3	INVALID_REQUEST	Request is Invalid
4	WEBAUTHN_NOT_ALLOWED_ERROR	User cancelled biometric prompt
5	USER_ABORTED	User clicked on skip or verify another way
6	ERR_BAD_REQUEST	Invalid command, some field is either missing or invalid

4.4 Error Codes

S. No	Error Codes	Error Classifications	Description
1	400	BAD_REQUEST	The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing).
2	401	UNAUTHORIZED	The user does not have valid authentication credentials for the target resource.
3	500	INTERNAL_ERROR	Internal Server Error, Ref Error descriptions and details for more information.

5 API Reference

For API Specifications, please reference separate API documentation.

6 Change Log

Version	Description	Date

0.1	Initial Draft	31 Jan 2025
1.0	External Version	07 Mar 2025
1.1	Edits to Improve Consistency	1 Aug 2025