# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>INTRODUCTION</td>
</tr>
<tr>
<td>03</td>
<td>SCOPE OF THE DOCUMENT</td>
</tr>
<tr>
<td>04</td>
<td>WHAT IS FINANCIAL-GRADE API SECURITY?</td>
</tr>
<tr>
<td>08</td>
<td>TECHNICAL REQUIREMENTS (AND CHALLENGES) OF OPEN BANKING</td>
</tr>
<tr>
<td>11</td>
<td>HOW PING USES IAM TO ADDRESS OPEN BANKING</td>
</tr>
<tr>
<td>20</td>
<td>CONCLUSION</td>
</tr>
<tr>
<td>21</td>
<td>APPENDIX: GLOSSARY OF TERMS</td>
</tr>
</tbody>
</table>
INTRODUCTION

The concept of "open banking" is currently pushing the envelope in the API security world. Mandates, including the PSD2 directive in Europe, the Open Banking Standard in the UK, the Open Banking Project in Germany and individual bank initiatives, are requiring financial institutions to open their APIs to allow access by third-party providers. How can these "open banking" APIs be secured against fraud and abuse?

The technology-neutral PSD2 doesn't specify the details of how ASPSPs should secure APIs. While the Competition and Markets Authority, responsible for the UK Open Banking Standard, has already specified the use of OAuth 2.0, exact technical specifications are still in flux.

In either case, unique snowflake implementations will not thrive in the large, trusted third-party ecosystems (consisting of fintechs, merchants and other financial applications) that we expect to emerge. To allow for secure, scalable implementations and the leveraging of highly specified, heavily examined identity and security patterns, adoption of industry-standard API security architectures is required.

SCOPE OF THE DOCUMENT

This solution guide expects that you are familiar with the basics of PSD2 and Open Banking, including the types of third-party providers (TPPs), Account Servicing Payment Service Providers (ASPSPs) and common use cases. To get up to speed on the particulars, read our eBook: PSD2 & Open Banking.

The purpose of this guide is to explain the basics of an OAuth-based, financial-grade security model. It describes how an ASPSP might issue tokens to a TPP securely and in such a way that the TPP could subsequently use those tokens at that ASPSP with minimal risk.

The guide does not define how actual APIs for payment and account information are constructed or what kinds of payloads or endpoint data might be present. In addition, security considerations and solutions for TPPs, such as how the TPP secures the access tokens, are out of scope for this guide.

STILL SETTING UP APIS? OUR PARTNERS HELP WITH THAT.

The first step to meeting requirements for PSD2 and Open Banking is constructing open APIs for payments, transaction data and other account info. Are you still looking for help with setting up financial-grade APIs in a portal/gateway for third-party access?

Ping's world-class partner network includes dedicated, specialized financial technology companies that serve banks. Many have created platforms or provide an API layer/portal, often as a managed service. These technology companies can accelerate getting the APIs you need in place, but securing them is still an important component—and that's Ping's primary role. Rather than build security and access management from scratch, many of these types of companies are using Ping's capabilities as the security component, either white-labeled or openly powered by the Ping platform.

Please connect with an account executive today to learn more about how we can leverage the right industry partners to provide the complete solution that your bank needs to comply with PSD2 and Open Banking.
WHAT IS FINANCIAL-GRADE API SECURITY?

OAuth has been in use for several years to secure access to APIs in a simple and standardized method. A common use case for OAuth has been to allow third-party consumer applications limited access to user data from another application, like Google or Salesforce.com, with end user consent. The value is simple, improved security. With OAuth, those third parties are only allowed scoped authorization, and credentials are never shared. But there’s a problem as well. Those use cases don’t come close to meeting the security requirements for financial-grade API security.

To solve for this problem, we recommend that the OAuth 2.0 family of specifications be used as the basis of any scheme to protect financial APIs. While not all open banking initiatives require the use of OAuth, there is no other specification family that combines all the elements needed for a distributed ecosystem of third-party RESTful API clients. OAuth is the best practice deployment option and will be the likely default implementation within and outside of mandated schemes.

Because the specifications shown above apply to both high and low assurance purposes, following the specification alone is not enough to ensure a financial-grade architecture.

To determine how the specifications should be used, consult the OpenID Foundation’s Financial API working group implementation drafts. They profile portions of the above stack such that only high assurance configurations are acceptable. The first draft of the read-only specification can be found here. The configured solutions in this guide are based on these FAPI profiles.

Reference Example

To illustrate these concepts, imagine a money management application wants to aggregate your banking habits across all of your financial institutions.

1. The end user (called a Payment Services User, or PSU) signs up at the Management of Money (MoM) app in the role of the third-party provider (TPP).
2. The PSU then indicates that they are customers at two separate ASPSPs: Royal Bank of Trafalgar and Worldly Bank.
3. Assuming that the PSU grants consent, and assuming that trust has been established in some way between MoM and the ASPSPs, MoM should be able to access REST APIs on behalf of the consenting PSU to show aggregated data and provide additional services based on the account information. Or alternatively, it allows the MoM app to initiate a payment request to Royal Bank of Trafalgar or to Worldly Bank.
Token-based API Authorization

Financial institutions that act as ASPSPs will offer a number of REST APIs for use by the TPPs. Each API will expect an OAuth 2.0 “access token” to be passed with every API call.

The access token is issued by the affiliated ASPSP and represents limited authorization for the TPP to interact with APIs on behalf of the payment services user (PSU). By using access tokens, you remove the need for the TPP to collect actual PSU credentials. You also remove the need for each ASPSP payment API to collect and validate user credentials. Instead, a short-lived token is issued and becomes common currency that can be requested by TPPs, issued by ASPSPs and passed to APIs hosted by the ASPSP to enable access.

In Figure 2, the MoM app would cache access tokens for both Royal Bank of Trafalgar and Worldly Bank, using each access token at the appropriate institution’s APIs to initiate payments or consume account information.

OAuth 2.0 and OpenID Connect Token Services

Access tokens can be requested from the ASPSP in multiple ways, but only the most secure methods should be used in financial situations. Code flow is the recommended OAuth flow for maximum security.

In code flow, the TPP uses the browser to make the request and to receive a short-lived “authorization code.” Once that code is received, the TPP makes a direct REST call to the ASPSP to retrieve tokens, authenticating to the token endpoint with strong client credentials. This model makes it difficult to steal tokens.

In Figure 3, if the MoM app didn’t already have an access token for Royal Bank of Trafalgar, or if the access token didn’t work, the MoM app would redirect Pam’s browser to Royal Bank of Trafalgar with an accompanying OAuth2 request. The MoM would receive back an authorization code that it would need to immediately use in a direct REST call to the Royal Bank of Trafalgar “token endpoint,” a resource that requires Royal Bank of Trafalgar to supply client credentials to authenticate. Of note, the MoM app has no visibility into how Royal Bank of Trafalgar authenticates Pam, or what kind of consent is collected from Pam.
Strong Customer Authentication (SCA)

PSD2 has specific requirements for how an ASPSP must authenticate users prior to API access being granted. The regulatory technical standard can be found [here](#). To summarize the regulations, the user must use at least two factors from two different categories of “things you know, things you have and things you are.” The exact UX and technical constitution of the authentication ceremony are otherwise considered to be owned by each ASPSP separately and proprietary to their needs.

Referring again to Figure 3, once the MoM app makes an authorization request to Royal Bank of Trafalgar, Royal Bank of Trafalgar may choose to prompt for a username and password and then to send a push notification to Pam’s trusted mobile device requiring Pam to take an action like a swipe.

Dynamic Linking

Because electronic remote payment transactions have a higher risk of fraud, they are subject to additional SCA requirements. Specifically, the authentication must dynamically link the transaction to the payment amount and the payee specified by the payer when initiating the transaction. This ensures the authentication can be traced back to the specific transaction and reduces the risk of “man-in-the-middle” attack where a bad actor changes the payee and/or amount after the payment has been authorized.

Consent

Consent is also considered to be independently managed and proprietary to each ASPSP. There is, however, guidance in various open banking and general data protection mandates about how granular the consent interface must be and how often consent must be requested.

Per the flow in Figure 3, once Pam is strongly authenticated, Royal Bank of Trafalgar might show Pam three online bank accounts she could consent to sharing with the MoM app. If Pam chooses only one, the MoM app never knows that the other two accounts exist and can never be used by the MoM app to retrieve information about the other two accounts.
Trust Establishment

It is possible that the MoM app negotiates trust agreements with Royal Bank of Trafalgar and separately with Worldly Bank and any other ASPSP they want to access. In mandated schemes like PSD2 and the Open Banking Standard, however, there is an authority that underwrites the digital trust between the ASPSP and TPP.

Instead of every ASPSP needing to establish trust with every TPP, they all enroll once with the authority. Once enrollment is complete, any TPP that is accredited by the authority should be able to register with any ASPSP accredited by the same authority. This model is designed to enable ubiquitous technical access.

As shown in Figure 4, once the MoM app is accredited at Open Banking R Us (the authority), they should have the capability to register at any ASPSP that is also accredited at Open Banking R Us using the same protocols and API patterns.

Figure 4: Example of an open banking authority underwriting digital trust between entities.
To map financial terms to technical roles, the ASPSP must provide infrastructure that acts in the role of an OAuth authorization server, and the ASPSP financial APIs must accept access tokens, playing the role of OAuth resource servers. The TPP is cast in the role of an OAuth client.

As shown in Figure 4, the MoM app is a client, while both Royal Bank of Trafalgar and Worldly Bank operate authorization servers and resource servers. It is expected that the MoM app will register with each authorization server before requesting access tokens on behalf of users.

The following technical requirements are based on the FAPI standards for financial-grade APIs, RFC 6819, the OAuth 2.0 Threat Model and research into different financial API architectures.

**Advanced Security Configuration**

To truly mitigate risk in OAuth 2.0, the privacy and security must be set to their highest levels. According to the FAPI workgroup, OAuth deployments would need to be configured in the following ways to meet financial-grade status:

- Endpoints are required to be protected by TLS 1.2.
- Secrets delivered to clients must have a minimum amount of entropy.
- Key sizes must be large (2048 bits or more for RSA, 160 bits for elliptic curve algorithms).
- Attempts to replay the authorization code should be detected.
- Access token minimum size should be 128 bits and content should be unguessable.

**Signed Authorization Request**

A signed authorization request is an extension of OAuth 2.0 that requires all the parameters in an authorization request to be passed to the authorization server as a single signed JSON Web Token (JWT) assertion. Because the parameters are signed by the TPP, the ASPSP can be confident they haven’t been tampered with.

Two specifications standardize the creation of signed authorization requests: the OAuth JWSreq draft and OpenIDConnect section 6.
Secure Client Authentication

Protecting secrets is the core of the OAuth security model, and client secrets are no exception. The three models currently identified as being secure enough for financial-grade transactions are as follows:

Direct Mutually Authenticated Client Authentication
- In this scheme, the client follows the [TLS Authentication for OAuth Clients spec](https://tools.ietf.org/html/rfc7635).
- The establishment of the mutually authenticated transport layer is supervised by the authorization server and identifies the client securely and uniquely.

JWT Assertion Bearer Authentication
- In this scheme, the client follows [RFC 7523](https://tools.ietf.org/html/rfc7523).
- Instead of a client password, a signed JWT is sent to identify the client. The authorization server validates the signature of the assertion with the public key of the client_id that was claimed in the authorization request.

Client Password with Mutually Authenticated TLS at Layer 3
- In this scheme, a client secret is used, but over a mutually authenticated TLS transport layer.
- Additional implementation logic is required to ensure that the established mutual TLS channel has credentials that match the client that claims to have authenticated with the password.

Code Verifiers

Mitigating the risk of authorization code theft by middle-man attacks in the browser is a critical security requirement for authorization servers. Proof key for code exchange (PKCE, also known as [RFC 7636](https://tools.ietf.org/html/rfc7636)) is likely to be required for most clients operating in financial-grade ecosystems, although it is particularly critical for public clients. It's likely that some financial ecosystems will not allow public clients at all, meaning that any TPP-ASPSP interaction will be required to be a server-to-server interaction. Use of PKCE is recommended wherever possible in current drafts of the [OAuth Security Topics BCP](https://tools.ietf.org/html/rfc7636).

Redirect URI Hygiene

How clients specify where tokens or codes should be delivered after they're generated is yet another area where risk mitigation is critical. The base OAuth specification doesn't heavily specify the need for tightly validated and pre-registered redirect URIs, but the OpenID Connect specification and the FAPI specifications make very specific recommendations:

- Redirect URIs must be registered in advance at the authorization server.
- The client must include the redirect URI as a parameter value in the authorization request.
- The specific redirect URI used in a given authorization request must exactly match one of the pre-registered values (not just the domain, but the entire value).
- Clients must maintain a separate redirect URI for each ASPSP.
- Clients must check that the token is sent to the redirect URI instructed by the ASPSP.
Advanced Context and Accountability

It benefits all parties to communicate information about the transaction and the security context of various interactions (excluding PII, of course, and maintaining privacy guardrails). There are multiple channels for communication of metadata:

Staging APIs

- A TPP may need to call an ASPSP API to register intent to transact before the TPP introduces the PSU at the authorization server. The ASPSP may then need to retrieve the staged information in order to understand the context for consent gathering purposes.

Signed Request Objects

- Context about a transaction could be passed as an attribute in a signed request object. In this case, any sensitive information would need to be encrypted.

ID Tokens

- Context about the user-consented transaction, the authentication context of the user or security context such as a hash of the issued access token can all be passed in an OpenID Connect id_token.

Headers

- The FAPI profile recommends that clients calling a resource server set a number of headers to establish the following context:
  - X-fapi-interaction-id: a tracking number for individual transactions that can be logged for troubleshooting purposes (set by the client)
  - X-fapi-financial-id: an identifier for the intended financial institution (useful if a client has been fooled into presenting a token at the wrong place)
  - X-fapi-customer-last-logged-time: a value describing the last time the customer authenticated to the client

Dynamic Client Registration and Discovery

There are three possible ways for a client to register at an authorization server:

1. **Manual**: An administrator uploads or imports certificate and client metadata from an individual email or an authorized metadata list.
2. **Self-service**: The developer of the TPP is given credentials to maintain their own client metadata at a portal built for that purpose.
3. **Automatic**: The client receives an initial client secret or assertion that entitles it to register itself at a registration endpoint (or possibly to register at multiple registration endpoints for multiple authorization servers).

In an ecosystem of massive scale, the manual and self-service options for client registration become impossible. In this case, authorities are likely to mandate that ASPSPs operate a standards-based registration endpoint, operating in compliance with RFC 7591 and RFC 7592. In addition, clients must have a way to discover the location of the registration endpoints for multiple ASPSPs. They are likely to either use OpenID Connect Discovery or the OAuth 2.0 Authorization Server Metadata draft.
Identity and access management (IAM) ensures the right people (and only the right people) can access the right applications, services and APIs seamlessly and securely. Providing optimal security, while delivering an equally strong user experience, IAM is perfectly suited to the challenges presented by open banking.

Ping’s IAM solution, the Ping Identity Platform, provides federated single sign-on (SSO), multi-factor authentication (MFA), access security, directory and data governance. To solve for the specific challenges of open banking, it can:

- Issue tokens based on strong customer authentication and consent.
- Store identity, policy and consent data necessary to issue properly scoped access tokens to TPPs.
- Facilitate dynamically linked customer authentication.
- Act at each API to validate and introspect the submitted access tokens, supplying necessary identity and scoping contexts to payments and accounts APIs.

Deployed individually or in combination, the Ping Identity Platform provides solutions to address the full spectrum of open banking requirements.

The **PingFederate** server can be configured to issue tokens from multiple protocol endpoints either singly or in nested configurations, including:

- OAuth 2.0 Authorization Server
- SAML 2.0 Identity Provider and Relying Party
- Productized integrations to proprietary solutions
- OpenID Connect Identity Provider and Relying Party
- SCIM outbound and inbound interfaces
- Custom integrations

The **PingAccess** server can be configured either as a proxy or an agent to offload the burden on APIs of validating access tokens and enforcing policy.

**PingDirectory** can act as highly available secure storage for user identities, consent records and policy information.

**PingDataGovernance** offers REST APIs for management of identity and authorization data, filtered by policy.

The **PingID** cloud MFA service plays a key role in strong customer authentication.

---

**Figure 5:** The Ping Identity Platform addresses the full range of open banking requirements.
Standards-based API Access

PingFederate is designed to work with custom and standards-enabled products of all types, including data stores, authentication services, API management tools, policy engines and target resources.

It fully supports TLS 1.2 and large key bit sizes, as well as RFCs 6749, 6759, 7009, 7636 (PKCE), 7662, 7521 and 7522, OpenID Connect Core and Discovery. Support for RFC 7523 is planned in a future release of PingFederate giving implementers the option to use JWT assertions for client authentication, and OpenID Connect section 6.0, which will give implementers the option of using signed authorization requests.

PingFederate operates as an OAuth 2.0 authorization server. It also operates additional proprietary APIs that enable PingAccess to act as a standards-compliant OAuth 2.0 client for the purposes of token validation, as well as an enforcement point with additional policy intelligence. This includes continuous authentication and step-up functionality.

As shown in Figure 6, PingFederate:

1. Operates an OAuth 2.0 authorization endpoint, where clients can make browser-based requests for access tokens, refresh_tokens, or id_tokens. OpenID Connect requests are also supported at this endpoint. Depending on the flow, PingFederate may return authorization codes, id_tokens, or access tokens. PingFederate uses authentication policies to link authorization requests to any kind of SCA ceremony - including login forms, MFA, SAML authentication requests, and custom integrations.

2. Operates an OAuth 2.0 token endpoint, where clients can make direct HTTP calls to request access tokens, exchange authorization codes for tokens, or refresh access tokens. PingFederate supports both mutual TLS authentication and client password authentication today, with JWT assertion bearer client authentication coming soon.

3. Operates standards-based token introspection and revocation endpoints that can be accessed by any standards-compliant API code or product.

4. Operates additional proprietary APIs that enable PingAccess to not only act as a standards-compliant OAuth 2.0 client for the purposes of token validation, but also as an enforcement point with additional policy intelligence, including continuous authentication and step-up functionality.

Figure 6: PingFederate operates as an OAuth 2.0 authorization server.
Strong Customer Authentication and Consent

While providing a frictionless user experience will become a critical differentiator, it’s up to each organization to develop its own user experience for authentication and consent. The Ping Identity Platform delivers customizable templates, plus provides extension points and APIs to allow for deep functionality to be executed. Solutions discussed in this section represent only a small subset of configuration possibilities.

Strong customer authentication with simple consent as shown in Figure 7 might work as follows:

1. The client attempts to access an open banking API protected by PingAccess.
2. Access token validation fails. The client makes an authorization request via the PSU browser to PingFederate for a new token. Authentication policies prompt the PSU via an HTML login form (asking a “what you know” factor).
3. User credentials supplied by the PSU are validated against the PingDirectory.
4. PingID is invoked to prompt the user to interact in some way with a trusted device (a “what you have” factor).
5. PingFederate displays a consent form directly linked to the scopes that the client has requested.
6. The PSU browser delivers an authorization code to the client. The client fetches an access token (exact detail not shown for the sake of simplicity).
7. PingAccess validates the access token and sets headers communicating identity and scope data to the open banking API.
8. The API request and subsequent requests are successful while the access token is valid.
A further explanation of the use case shown in Figure 8 is as follows:

1. The client attempts to access an Open Banking API protected by PingAccess.
2. Access token validation fails. The client makes an authorization request via the PSU browser to PingFederate for a new token. Authentication policies prompt the PSU via an HTML Login Form (asking a "what you know" factor).
3. User credentials supplied by the PSU are validated against the PingDirectory.
4. PingID is invoked to prompt the user to interact in some way with a trusted device (a "what you have" factor).
5. PingFederate redirects to a custom consent application, where the user specifies detailed consent that goes past simple scope-based agreement.
6. Consent granted by the PSU is stored in PingDirectory and communicated to PingFederate.
7. PingFederate delivers an authorization code to the client via the PSU browser. The client fetches an access token (exact detail not shown for the sake of simplicity).
8. The client attempts again to access the open banking API. PingAccess validates the access token and sets headers communicating identity and scope data to the API.
9. The API request and subsequent requests are successful while the access token is valid.

Figure 8: A sample use case showing strong customer authentication with customer app-driven consent.
Strong Customer Authentication with Dynamic Linking

1. The client attempts to access an Open Banking API protected by PingAccess.
2. Access token validation fails. The client makes an authorization request via the PSU browser to PingFederate for a new token. Authentication policies prompt the PSU via an HTML Login Form (asking a "what you know" factor).
3. User credentials supplied by the PSU are validated against PingDirectory.
4. PingFederate redirects to a custom consent application, where the user specifies detailed consent that goes past simple scope-based agreement.
5. Based on the consent provided, PingFederate composes a detailed, dynamically-linked transaction approval message. PingFederate interacts with PingID to send the approval message to the PSU’s mobile device.
6. The PingID SDK, which is incorporated into a custom app provided by the ASPSP, displays the approval message and requires the PSU to confirm the transaction via fingerprint impression or some other positive action.
7. On confirmation of this second-factor approval, consent granted by the PSU is stored in PingDirectory and communicated to PingFederate.

OAuth 2.0 API Protection

Once the TPP client has an access token, the access token can be passed as an authorization header to Open Banking resources. PingAccess offloads the critical security task of the validation, introspection and enforcement of OAuth 2.0 access tokens. It also adds an additional policy layer that can act on changes in environmental context at the time the client calls the resource, resulting in the ability to make event-driven medium-grained authorization decisions.

PingAccess can perform in different capacities for different APIs, acting as a proxy to the API or acting as an external agent referenced by the API, but not in the traffic flow.

PingAccess as a Proxy

Figure 9: PingAccess can act as a proxy to the API.
PingAccess acting as a proxy (as shown in Figure 9) works as follows:

1. The TPP client uses an already requested access token to make an API call.
2. PingAccess (acting as the API endpoint) pulls the access token from the authorization request and calls PingFederate.
3. PingFederate validates the access token and retrieves relevant identity and scope information.
4. Identity, consent and scope information from PingFederate are used in conjunction with environmental context information to evaluate access policies.
5. The request is forwarded to the application (passing relevant data).
6. The API completes the transaction and returns the result to the client.

**PingAccess as an Agent**

![Diagram of PingAccess as an Agent](image)

*Figure 10: PingAccess acting as an external agent referenced by the API.*

When PingAccess acts as an external agent (as shown in Figure 10), the flow looks like the following:

1. The TPP client uses an already requested access token to make an API call.
2. The API itself (or a container for the API, such as NGINX, Apache or F5) acts as the API endpoint, pulls the access token from the authorization header and calls PingAccess.
3. PingAccess calls PingFederate.
4. PingFederate validates the access token and retrieves relevant identity and scope information.
5. Identity, consent and scope information from PingFederate are used in conjunction with environmental context information from the API to evaluate access policies.
6. The request is forwarded to the application (passing relevant data).
7. The API completes the transaction and returns the result to the client directly, rather than through PingAccess.
Trust Establishment

PingFederate offers a client management API that is not standardized. It is intended only for privileged clients within the ASPSP. This internally facing API can be called either by a custom self-service developer portal, by an API management portal with a PingFederate integration in place or from an externally facing automated interface compliant to RFC 7591 and RFC 7592. Exact implementation and the security involved in authorizing either developers or clients is custom designed to user requirements.

A sample use case (as shown in Figure 11) is as follows:

1. The TPP registers their application's metadata at an open banking authority and receives a software statement (a signed JWT assertion)* in return.
2. The TPP client calls a PingAccess-protected custom RFC 7591 interface, submitting the software statement as authorization to access the interface.
3. The software assertion is validated according to custom requirements from the authority.
4. On successful validation, the client metadata is submitted by the RFC 7591 interface to the PingFederate client management API.
5. A client is created, updated, or deleted in the PingDirectory.

*Software statements may be used when an authority is responsible for transitive trust between the TPP client and the ASPSP. This is explained in RFC 7591.

Establishing Trust with PingFederate

Figure 11: Use case illustrating how the Ping Identity Platform manages trust establishment.
Payments with Two-stage Commit

Payments via a two-stage commit process represents a prevalent real-world use case for open banking. In this case, the TPP acts specifically as a Payment Initiation Services Provider, or PISP.

Referring back to our previous examples, imagine that Pam has decided to purchase a $100 yearly subscription to the Management of Money App and wants to pay for it using her Royal Bank of Trafalgar savings account. In this case, the MoM app will initiate a payment, requesting that Royal Bank of Trafalgar pay MoM Inc. $100.

This entails a three-step process:

1. MoM Inc. “stages” the transaction with Royal Bank of Trafalgar, informing Royal Bank of Trafalgar of details such as the amount, the Payee (MoM Inc), the currency, etc.
2. The MoM app redirects Pam to Royal Bank of Trafalgar to consent to the payment and receive an access token.
3. The MoM app initiates the payment on behalf of Pam at the Royal Bank of Trafalgar payment API.

Ping Identity Platform: Two-stage Payment with Commit

Figure 12: The Ping Identity Platform addresses payments with two-stage commit employing best practices as outlined in the FAPI recommendations.
This common use case demonstrates how the Ping Identity Platform can address all of the best practices outlined in the FAPI recommendations:

1. Pam initiates the payment in her MoM app, selecting Royal Bank of Trafalgar as the payment vehicle, which results in a call to the MoM Inc. servers.
2. MoM Inc. makes a REST call to the Royal Bank of Trafalgar staging API to create a pending transaction record. Royal Bank of Trafalgar returns a transaction id (note that no user is associated yet).
3. The MoM app opens a browser (conforming to the OAuth for Native Apps BCP and FAPI recommendations) and redirects to the MoM Inc. servers, asking for a MoM Inc access token.
4. MoM Inc. servers redirect to the Royal Bank of Trafalgar ASPSP making an OAuth 2.0 authorization request to PingFederate, via mutually authenticated TLS with a PKCE code challenge included as well as passing the transaction id retrieved from the staging API.
5. PingFederate strongly authenticates Pam:
   a. Prompts Pam to enter her username and password.
   b. Validates the entered credentials against the directory.
   c. Calls PingID service which:
      d. Prompts Pam to use Touch ID to verify a fingerprint on her iPhone.
6. Pam is prompted to consent to make the payment (consent record is stored in PingDirectory) and to choose an account. The payment amount in the consent message is fetched by de-referencing the transaction ID passed in the authorization request.
7. PingFederate issues an authorization code to the MoM Inc. client.
8. The MoM Inc. client fetches the access token directly without any browser involved, using its pre-registered client id and secret over a MATLS-protected transport layer. PingFederate validates the PKCE code verifier before returning the tokens.
9. The MoM Inc. server caches the access token and calls the payment API with the access token to initiate the staged payment. The Royal Bank of Trafalgar token is never sent to the MoM app.
10. MoM Inc. servers return an access token or id_token to the MoM mobile app allowing the mobile app to see status of the payment as filtered by the MoM Inc. servers. The MoM app is unable to directly access Royal Bank of Trafalgar payment APIs.
CONCLUSION

Open banking holds the potential to radically change the consumer financial landscape. By breaking down traditional competitive barriers, it opens up significant opportunities for those who can quickly adapt.

But given the sensitive nature of the information being accessed, compliance alone is not sufficient. Solutions must be centered on an architecture that embodies the state of the art in API security. And to stand out in an increasingly competitive environment, the user experience must not be compromised.

OAuth 2.0—when profiled according to recommendations from industry experts such as the OpenID Connect Financial API Working Group—can provide the security needed. And when employed as part of a comprehensive IAM solution, it can deliver the optimal balance of security and user experience.

The Ping Identity Platform is designed to solve the exact technical challenges that open banking creates. Built on open standards, its advanced integration capabilities position you to be ready for anything.

From PSD2 and Open Banking Standard compliance today, to new requirements as the technical and regulatory landscape evolves, you can rely on the Ping Identity Platform to provide a solid foundation for a highly secure API framework that delivers a frictionless user experience and distinct competitive advantage.

To learn more, visit www.pingidentity.com/psd2.
APPENDIX: GLOSSARY OF TERMS

Access Token, Refresh Token
Credential(s) issued to a client that entitles the client to interact with RESTful APIs. The client treats the token as opaque, meaning that the client should not expect to read or interpret the token, only to pass it on.

ASPSP (Account Servicing Payments Service Provider)
An entity in the banking ecosystem that services customer accounts. Examples of ASPSPs are large financial institutions, like Royal Bank of Scotland or Nationwide. An ASPSP authorizes user/client interactions, protects APIs, supports discovery and dynamic client registration.

Authorization Code
A very short-lived ephemeral secret passed to a third-party provider client over the browser. The authorization code is meant to be instantly traded over a non-browser channel for an access token.

MATLS (Mutually Authenticated TLS)
A method of transport layer security where both sides of the connection must provide Layer 3 credentials.

PISP (Payment Initiation Services Provider)
A specific role that a third-party provider can play where it can initiate a payment transaction with an ASPSP.

PKCE (Proof Key for Code Exchange)
Standard securing the OAuth token endpoint from authorization code theft.

SCA (Strong Customer Authentication)
Term for a multi-factor authentication ceremony used in PSD2 and described in the PSD2 Regulatory Technical Standard.

TPP (Third-party Provider)
An entity in the banking ecosystem that wishes to interact in some way with open banking APIs. Examples of TPPs might be fintechs, merchants or apps that consume banking data, like Mint or Yodlee. Large financial institutions may play both the role of ASPSP and third-party provider against other large institutions. TPPs act as OAuth clients, requesting tokens in order to access open banking APIs.