Flexible Authentication

This document notes down the APIs to be called in what sequence for below mentioned basic FIDO operations which includes the new feature of Multiple Assertions Processing and Flexible Authentication:

1) FIDO Registration

2) FIDO Authentication

**RB and FDI** Decides on Authenticators supported by device on which the PSL Bank app is installed.

**A] FIDO Registration flow:**

RB

FDI

FIDO Client App

PSL Bank App

FIDO Server

PSL Bank Server

 



















 

**Initial Pre-Requisite step:**

User installs the FIDO Client app and on opening the app the first time, accepts the pop up for push notifications. This fires below API on FIDO server:

*Description*: Allow Push notifications when user installs the FIDO client app and runs it for the first time.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/enablePushNotifications

*Method*: POST

*Input*:

{

"deviceid": "0000111111",

"devicetoken": "hksdcvkdfjcvnhdfdfdfdfdfdfdfdfdfdfdfdfdfdfdfdfvcsdfvvvvvvvvvvvvvvvvvvvvvvvvvs"

}

*Output*:

{

"deviceDataUpdateStatus": "true/false"

}

**FIDO Registration Flow: (Please refer above block diagram)**

**STEP 1:**

User logs into the PSL Bank App using the traditional login (If user is not registered with PSL Bank app, then user will have to register himself before doing this step.), goes into settings and toggles the FIDO Registration option to enable registration with FIDO for the logged in user account. This details is in-turn forwarded to the PSL Bank backend server for storage and further processing.

**STEP 2:**

The PSL Bank server makes a call to the FIDO server API by passing the required user’s account details and the different types authenticators present in the world with help of RB (Risk Based management System) and FI (Federated Identity System), as JSON Input object. Below API is called in this case:

*Description*: APP Registration Request.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/appRegRequest

*Method*: POST

*Input*:

{

"rpDisplayName": "HDFC",

"displayName": "Amogh",

"email": "amogh\_tarcar@persistent.com",

"accountId": "11123"

}

*Output*:

{

"registrationResponse": "688664"

}

**STEP 3:**

Amogh: FIDO server processes the request and makes sure it supports these AAIDs

FIDO server processes the request and returns back an OTP (One time Password) to the PSL Bank Server.

**STEP 4:**

The PSL Bank server forwards this OTP to the PSL Bank App which displays it as a QR code.

**STEP 5:**

PSL Bank App in-turn forwards the control to the FIDO client app by passing the OTP details and the user account details to it.

**STEP 6:**

FIDO Client app scans the QR code displayed and passes the scanned information along with the deviceid back to FIDO server. This fires below API:

*Description*: Verify QR code contents after scanning.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/verifyQRContents

*Method*: POST

*Input*:

{

"rpDisplayName": "",

"displayName": "",

"email": "",

"otp": "",

"accountId": "",

"deviceId": ""

}

*Output*:

{

"QRCodeVerification": "true/false"

}

**STEP 7:**

FIDO server processes the request and returns as true or false depending on whether the QR contents were verified or not to the FIDO client app.

**STEP 8:**

The FIDO client app, next fires the FIDO regRequest API as below:

*Description*: Reg Request API that returns a policy containing the allowed and disallowed set of authenticators

*URL*: http://localhost:8080/fidouaf\_test/v1/public/regRequest/{rpAccountId}

*Method*: GET

*Output*: Challenge, Username, APPID, serverData, Policy, allowed/disallowed list of authenticators are all returned together as a JSON. ***[Refer FIDO Protocol document for the same]***

**STEP 9:**

FIDO server processes the requests and sends back a policy containing allowed and disallowed set of authenticators to the FIDO Client app. [*NOTE*: *This list of allowed authenticators will be the same that were sent during STEP 2. For now these values for AAIDs are hardcoded in the code and are nowhere stored in MYSQL DB*.].

Amogh: Step 10 will not be needed as the FIDO Client scans the policy and tries to find a match locally on the device. No-interaction is needed with the PSL Bank. The policy includes only those authenticators which were initially sent by PSL bank server to FIDO server.

**STEP 10**:

FIDO client app in-turn forwards this list to PSL Bank app which consults its PSL bank backend server to check if the device supports any or all out of the list of allowed authenticators and replies back with the supported authenticators back to FIDO Client app.

**STEP 11:**

~~Based on the list of supported authenticators out of the list of allowed authenticators as per the policy, received from the PSL Bank app~~, the FIDO client app builds assertions for each of the supported authenticators that it plans to get FIDO registered. This set of assertions are then forwarded to the FIDO server for processing. This causes below API to be fired:

*Description*: Reg Response API that processes assertions for each authenticators and stores the public key for each of them in FIDO server DB.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/regResponse

*Method*: POST

Input: Array of all assertions, assertion scheme, server data, APP Id and other related important fields. ***[Refer FIDO Protocol document for the same]***

*Output*: Returns status for each assertion whether SUCCESS or FAILURE along with AAID, KEY ID, Public Key and other important fields. ***[Refer FIDO Protocol document for the same]***

**STEP 12:**

FIDO server processes each assertions, stores the AAID, KEY ID and public key for each authenticator in DB and returns back a JSON giving status as SUCCESS of FAILURE for each of this authenticator based on whether they were registered successfully or not to the FIDO client App.

Amogh: FIDO Client application interacts only with PSL Bank App. All interactions of PSL Bank app and its server are independent. PSL bank app will send the list of registered authenticators to its server

**STEP 13 and 14:**

The FIDO Client APP then informs the PSL Bank App and its PSL Bank backend server about the authenticators that were successfully registered by FIDO server. This causes below API to be fired on PSL Bank server side:

***NOTE: FIDO client will make this call at the end of FIDO registration process, after FIDO server has sent reply to regResponse API as SUCCESS for each authenticator. This is to inform the PSL Bank App’s backend server about the authenticators registered and store the AAIDs for those authenticators in DB on PSL Bank server side.***

*Description*: API to inform PSL BANK server about the authenticators that are FIDO registered via FIDO client.

*URL*: http://localhost:9090/RelyingParty/public/registeredAuthenticators

*Method*: POST

*Input*:

{

"accountId": "111222",

"aaids": [

"PIN0#0001",

"TCH0#0001"

]

}

*Output*:

{

"status": "SUCCESS/FAILURE"

}

**END NOTE:** Above is the sequence followed for FIDO registration via Inter-app for flexible authentication feature. This completes the Registration flow.

**B] FIDO Authentication flow:**

**RB and FDI** Decides on Authenticators supported by device on which the PSL Bank app is installed.

RB

FDI

FIDO Client App

PSL Bank App

FIDO Server

PSL Bank Server



 



 







 

 

**15**

**16** 

**17**

User is now FIDO registered and can use the newly FIDO registered authenticators set based on the context. We have considered Location as the user’s context for now, and we can have pin-code based and fingerprint based authenticators developed for our App. Below are the authenticators enforced for different Locations beforehand:

|  |  |
| --- | --- |
| **Context** | **Authenticators (AAIDs)** |
| Home | PIN0#0001 |
| Office | TCH0#0001 |
| Other | PIN0#0001,TCH0#0001 |

Where, PIN0#0001 = Pin-code based Authenticator AAID and

TCH0#0001 = Fingerprint based Authenticator AAID.

**FIDO Authentication Flow: (Please refer above block diagram)**

**STEP 1:**

User logs into the PSL Bank App using FIDO. On click of Login button it will fire below API on PSL Bank server, which will return the authenticator AAIDs allowed for the sent user context based on above table(In future logic will be stored in Risk Based and Federated Identity Management systems). It will also internally call a FIDO server API to notify the server about the AAIDs being enforced on the selected context of the user:

*Description*: Authentication Request to get the Authenticators to be used for logging in.

*URL*: http://localhost:9090/RelyingParty/public/authRequest

*Method*: POST

*Input*:

{

"accountId": "111222",

"location": "Home/Office/Other"

}

*Output*:

{

"location": "Home/Office/Other ",

"authenticatorsAllowed": "PIN0#0001, TCH0#0001",

"fidoNotificationStatus": "notified/ error\_in\_update/ error "

}

Internal FIDO server API which will be called from within the above API is:

*Description*: Notification Request to FIDO server to notify about the authenticators enforced during login in based on user context.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/notifyAuthenticators

*Method*: POST

*Input*:

{

"rpaccountId": "111222",

"context": "Home/Office/Other",

"authenticators": [{

"aaid": " PIN0#0001"

}, {

"aaid": " TCH0#0001"

}]

}

*Output*:

Output is a string which can take values: NOTIFIED/ error\_in\_update

**STEP 2:**

PSL Bank Server will reply with the authenticators to be used for sent context for the given user back to the PSL Bank App.

**STEP 3:**

PSL Bank Server will also internally call an API on FIDO server sending the list of authenticators selected for logging in to make it aware/notify. **Refer STEP 1 for this API details.**

**STEP 4:**

FIDO Server will reply back to the PSL Bank server as “notified” after noting down (storing) the authenticators enforced for the said user context during login.

**STEP 5:**

Next, PSL Bank APP, forwards the details required to make the FIDO server’s authRequest API call to the FIDO client App. This details will include information such as AccountId, the authenticator AAIDs selected and the Context.

**STEP 6:**

FIDO Client App makes the FIDO server’s authRequest call by passing the AccounId of the User and gets the policy created in return. This policy JSON will include only those AAIDs which are enforced and are to be used during authentication. This causes below API to be fired:

*Description*: Authentication Request to FIDO server.

*URL*: http://localhost: 8080/fidouaf\_test/v1/public/authRequest/flexiblePolicy/{accountid}

*Method*: GET

*Output*:

As Expected. The Policy JSON will include only those authenticator AAIDs which were selected and returned during STEP 1 above.

**STEP 7:**

FIDO server returns back a Policy JSON as output to the AuthRequest call which contains the AAIDs that were selected during STEP 1.

**STEP 8:**

Based on Authenticator AAIDs returned in policy JSON, the FIDO client app makes the assertions for each authenticator.

**STEP 9:**

FIDO Client App, next makes the authResponse call and passes the assertions for each authenticators to the FIDO server for processing. API is the same as before. No change was done to this API:

*Description*: Authentication Response to FIDO server.

*URL*: http://localhost: 8080/fidouaf\_test/v1/public/authResponse

*Method*: POST

*Input*: As Expected.

*Output*: As Expected.

**STEP 10:**

FIDO server processes each assertion and verifies it using the public key stored for each of that authenticator. It replies back with a JSON containing the SUCCESS status for each assertion verification back to the FIDO client App.

**STEP 11:**

FIDO Client App then forwards this reply to PSL Bank App to notify it that the assertions were successfully processes and verified by the FIDO server.

**STEP 12:**

PSL Bank App in turn notifies its PSL Bank server about the assertion processing and verification success.

**STEP 13:**

The FIDO client app, next asks for a signed context data from FIDO server, which is privately signed by him, to forward it to PSL Bank app to cross verify it using the public key. This step is to ensure that the FIDO server has himself done the assertion processing and verification and not by some imposter. A separate SSL key pair is generated solely for this purpose by the FIDO server and the public key is already shared and stored by the PSL Bank server, for future communication processing between the PSL Bank server and the FIDO server. The below API gives the signed context from FIDO server:

***Note: This API is fired by the FIDO client and not the PSL Bank Server.***

*Description*: FIDO Client Get Signed Context Request to FIDO server.

*URL*: http://localhost:8080/fidouaf\_test/v1/public/getSignedContext

*Method*: POST

*Input*:

{

"rpaccountid": "151"

}

*Output*:

{

"signedContext": "sIUt4WF079hKh7W/jjXJQ2c+x79t73fdDcQoD9adS8AD58SCx65LmTCc973RyXbPw//mJbNrgsSZSW6hZKRBbBM8lGu2Gou6Kz+XcOPQkAt/5W9ys5XsrmWhdHq7GnTPK4h15C4Lyj8908BgCSaxpGTI0Oq7xj1Rdkt3dIaKKhYHrLEYfvxPFIm795GcwzXUK7r3cO/GRl6bEryCmASGU3GkcO/0sxTZ0SAK7FmQNQO2FU4wqe7D8VDR+6amSUrExSY0XRKDvyjYkawrml7R7729guQm3Wrqw8roVtL8BH6vbDcih4rNyQ9JFPsGscqKxeDlEjpbYSjkiz8zHAK5uQ=="

}

**STEP 14:**

The FIDO server processes the get signed context request from FIDO client, and signs the context for the given accountid using the private key and sends it as an output to this API, back to FIDO client.

**STEP 15:**

The FIDO Client send this signed context data back to PSL Bank App for verification using public Key.

**STEP 16:**

The PSL Bank App, in turn sends this data along with the user’s account ID to its Backed PSL Bank Server for verification using public key stored on PSL Bank server side. This causes below API to be fired:

*Description*: Verify signed context Request to get the privately signed context data and verify it using public keys shared by FIDO server exclusively for this purpose.

*URL*: http://localhost:9090/RelyingParty/service/public/appContextVerification

*Method*: POST

*Input*:

{

"accountId": "151",

"signedContext": "sIUt4WF079hKh7W/jjXJQ2c+x79t73fdDcQoD9adS8AD58SCx65LmTCc973RyXbPw//mJbNrgsSZSW6hZKRBbBM8lGu2Gou6Kz+XcOPQkAt/5W9ys5XsrmWhdHq7GnTPK4h15C4Lyj8908BgCSaxpGTI0Oq7xj1Rdkt3dIaKKhYHrLEYfvxPFIm795GcwzXUK7r3cO/GRl6bEryCmASGU3GkcO/0sxTZ0SAK7FmQNQO2FU4wqe7D8VDR+6amSUrExSY0XRKDvyjYkawrml7R7729guQm3Wrqw8roVtL8BH6vbDcih4rNyQ9JFPsGscqKxeDlEjpbYSjkiz8zHAK5uQ=="

}

*Output*:

{

"status": "VERIFIED"

}

**STEP 17:**

The PSL Bank server app, gets the signed context data and verifies it using the Public Key stored and replies back to the PSL Bank App as VERIFIED/NOT\_VERIFIED.

**END NOTE:** Above is the sequence followed for FIDO Authentication via Inter-app for flexible authentication feature. This completes the Authentication flow and also ensures that authentication is processed and validated by FIDO server himself via the sign and verify context APIs.