POC on WebAuthn as an auth factor in eSignet (Draft)

1. INTROUDCTION

In these times of digital transformation, most services are moving online globally. Personalized access to online services is enabled through the use of a trusted digital identity. **e-Signet** aims to offer a simple yet powerful mechanism for end users to identify themselves in order to avail of online services and also share their profile information. e-Signet supports multiple modes of identity verification to promote inclusion and increase access, thus narrowing potential digital divides.

e-Signet also provides an elegant and easy way for an existing trusted identity database to make the identity digital and provision identity verification and service access.

MOSIP eSignet is standard based authentication service interface which can work with any authentication provider.

Please refer to MOSIP eSignet documentation for more information.

eSignet implements OpenID Connect and OAuth 2.0 flows.

Platform today supports few authentication mechanisms such as biometric , OTP and Wallet. We would like to extend it to WebAuthn.

WebAuthn (W3 Web Authentication: An API for accessing Public Key Credentials - Level 3)

Key Information	Value
Complexity Level	Medium
Team Size	3-4
Timeline	Not more than 45 days
Tech Stack	Java/Springboot/REST/JSON React/node W3C Web Authentication OpenID Connect, OAuth 2.0, MDS
eSignet Target version	Branch 1.2.x

2. MOTIVATION

- OTP over email requires to have email access and could be delivered with a delay. Many a times OTP is delivered after expiry time which is not so comfortable experience for the user. SMS requires Telco/Aggregator support and adds to cost.
- WebAuthn implementation would enable the creation and use of strong, attested, scoped, public key-based credentials.
- WebAuthn is more secure mechanism since it uses Secure Execution Environment, Trusted Platform Module (TPM), or a Secure Element (SE) to store the keys
- Hence we need a solution which implements WebAuthn in eSignet.

3. CURRENT ARCHITECTURE

Please refer Components for details on the current architecure.

4. REQUIREMENTS & SOLUTION APPROACH

4.1 Requirement

Implement WebAuthn feature as per W3C WebAuthn in MOSP eSignet as explained below.

4.2 High level Solution Approach

4.2.1 Binding of new webAuthn credential with a specific user's identity

- Create a new spring boot based application for enabling webAuthn credential - User identity binding (** we keep this as new application, because it will guard from polluting the eSignet ui and server code to achieve this functionality)
- Create a UI page that offers users to enable webAuthn based authentication support to their trusted digital ID
- In the UI page, request the user to Sign In using their trusted ID
- Login process should follow the OIDC flow where this new app become one of the relying party
- Once user logs in, display a button to initiate webAuthn registration
- On user click, call the generate challange API endpoint in new spring boot application to receive a long random string.
- Call the Web Authentication API passing all the required information along with random string to find the available webAuthn authenticator and initiate creation a new keypair. This API requires a randomly generated string from server to avoid the replay attacks.
- On receive success response, convert the response to JSON and send the same to the new spring boot application for validation
- On successful validation of response at the server side, call the new webAuthn binding API endpoint of the eSignet application passing the clientDataJSON and authenticatorData from the response object (This new method invocation will be based on the access token)
- This new endpoint services needs to be part of the binding service where it has new method to do direct key bindings, which will validate all the required things as per the webAuthn specification
- This new binding service method calls the new method in the key binder plugin interface which has a new method for doing public key based binding (this method invocation will be passing the kyc token)
- The mock plugin impl will implement this new method where it will encrypt and store the public key along with credential id in the same record where the identity information is stored

4.2.2 Enhance eSignet to support the binded WebAuthn Credential as an authenticator factor

- eSignet UI to be customised to add a button in the login page to support "Login using WebAuthn Credential"
- On click of that we should show the login page which will request for individual id and show a button to initiate WebAuthn based authentication
- Use the transaction id that is already available in the eSignet UI as the challange and make the allowCredentials as empty and call get method on navigator.credentials with options - Reference doc
- The response object received from above get request is converted to the JSON string added as a challenge and with authFactorType as webauthn in authentication request and sent to eSignet server
- This request to the authenticate API endpoint and it will reach the plugin and then finally the mock ID system. (except for some enum and config changes there should not be much code changes here)
- Mock ID system will reach to its DB, get the key, decrypt and will validate the challenge response received. Credential id will be also be cross checked to ensure the credential id is same as that was received during binding

4.3 Modules & APIs to be developed or refactored

4.2.1 New Modules to be developed

4.2.1.1 WebAuthn Binder Application

- Build an Web Application using Java Spring boot RESTFul API Service and a React JS Front end application which connects to this backend. This application is Resident facing and allows them to bind their trusted ID (ex: UIN/VID) with a newly generated webAuthn credential generated in a residents device.
- A mock application developed in Node and React is available here (Appendix)
- In landing page following links/buttons should be provided
  - A link should be provided for Sign in with e-Signet (esignet-login-link) - Plugin can be used
  - A link/button for initiating creation of webAuthn credential and binding it to the identity (This link should be enabled only if login is successful) (generate_webauthn_credential_link)
- Need to implement the following UseCases
- **Login Flow**
  - login flow similar to the above mock implementation calling eSignet login UI . User will input trusted ID (UIN/VID). He may choose OTP based authentication.
  - Once user clicks esignet-login-link, login web-ui should appear from eSignet portal
  - Please consider only OTP based login for this project
  - Once login is successful control returns to application along with authorization code
  - Call /token API to send authorization code and and receive the access token
    - Access token is generated by eSignet service and saved in cache against a computed hash of access token.
    - This API will return access token
  - You may refer to this document to understand the flow and UI
  - We would require email ID as Essential claims
- **Create and bind webAuthn credential**
  - On click of generate_webauthn_credential_link, generate a random challenge from server side, use it to initiate new webAuthn credential generation
  - Resident will get the necessary user interations from the browser side to select the authenticator and generate key pair and respond back
  - Response received is sent to the server to complete the binding process
  - Resident is presented with a success message

4.2.2 New APIs to be developed

- **WebAuthn-binding**
  - Build a new webAuthn Binder API in ***eSignet/eSignet-services***, ***eSignet/binding-service-impl***
  - Add new API code in ***keyBindingController*** , ***KeyBindingServiceImpl*** similar to wallet-binding API.
  - This API takes in eSignet Access token , clientDataJSON and authenticatorData from webAuthn credential generation response and trusted Id (individual id) as parameters in addition to other required parameters
  - clientDataJSON and authenticatorData to be taken in same format as it was received in webAuthn credential generation response
  - Add a method bindWebAuthnCredential in ***KeyBindingServiceImpl*** class
    - Validate access token against cache
    - Validate all the other necessary things in the clientDataJSON and authenticatorData specification
    - Exact the publicKey and credential Id
    - If all validation are success, then save the binding along with credential id in registry as in bindwallet method

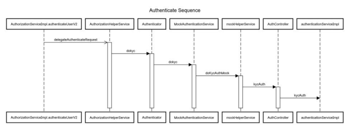
  - Add a new overloaded method to KeyBinder doKeyBinding method to take in kycToken instead of challengeList
  - Call the KeyBinder overloaded doKeyBinding method passing the webAuthn public key in the publicKeyJWK argument
  - Implement the new doKeyBinding method in MockKeyBindingWrapperServiceto call the new API endpoint in mock identity system
  - This new endpoint (key-binding) in mock idenity system should have a new controller and service layer to finally encrypt and store the public keys in the databases with a mapping to the identity
  - return success/failure

4.2.3 Modules to be refactored

4.2.3.1 eSignet-UI (eSignet/oidc-ui)

- Add new mode of login ***Login with WebAuthn*** to Login Page
- On selection of this option navigate to a new page to login using WebAuthn
- Here prompt for Individual ID
  - User would enter his/her trust id
- Provide a **verify** button (verify_link), and on click of this , call modified Authenticate API along with WebAuthn credential
- On return from this API, implement the consent flow as usual

4.2.3.2 Authenticate End point (Authentication Endpoint V2 | Identity Provider


Authenticate Sequence
AuthorizationServiceImpl.authenticateUserV2
AuthorizationHelperService
Authenticator
MockAuthenticationService
mockHelperService
AuthController
authenticationServiceImpl
delegateAuthenticateRequest
dokyc
dokyc
doKycAuthMock
kycAuth
kycAuth

-

Class Name	Repo
AuthorizationServiceImpl	https://github.com/mosip/oidc-service-impl/ Connect your Github account
AuthorizationHelperService	https://github.com/mosip/oidc-service-impl/ Connect your Github account
Authenticator - Interface	https://github.com/mosip/esignet/tree/1bc5ea2d4e8912dd782bcb79cfe40f2d1b5806ac/esignet-integration-api Connect your Github account
MockAuthenticationService	https://github.com/mosip/esignet-mock-services/tree/master/mock-esignet-integration-impl Connect your Github account
MockHelperService	https://github.com/mosip/esignet-mock-services/blob/master/mock-esignet-integration-impl
AuthController	https://github.com/mosip/esignet-mock-services/tree/master/mock-identity-system Connect your Github account
AuthenticationServiceImpl	https://github.com/mosip/esignet-mock-services/tree/master/mock-identity-system Connect your Github account

- Follow the flow and modify the parameters to add/modify if required
  - Mock implementation in esginet-mock-services , in method kycAuth of AuthentionServiceImpl class, default implementation fetches already generated with otp and compares with passed otp value.

    - Need to change the above logic to implement webAuthn validation

4.4 Development Environment

- Developers could fork from MOSIP gitrepo, all required Repos of specified branch and develop/debug locally
- Refer to Local Deployment guide for more details, which also covers how to insert test data (identity, partner details etc) using curl command.
- One could run the all necessary apps locally with custom implementation of webAuthn and test locally
- To test locally , one could use WebAuthn supported browsers like Chrome.
- Prepare test data & scripts to import into local database as per Local Deployment Guide

4.5 Deployment , Testing & Submission

- Submission shall be done by committing artefacts into the forked respective repo and raising pull request
- MOSIP Dev-ops team will verify, build and deploy to test sandbox and provide access to dev team
- Dev Team runs the tests and provides the report

5. DELIVERABLES

- All required Repos are forked from MOSIP Git Repo
- Raise pull request for all the customised code
- Check in the test scripts/cases, test reports
- Demo videos showcasing the working use-cases
- Any relevant documentation
- Deployment Script
    - Develop Docker composer scripts for local deployment of all services and running test scripts if any
    - Submit the same as part of deliverables
- Test data & scripts

6. APPENDIX

6.1 Reference Repos & URLS

6.1.1 MOSIP github

https://github.com/mosip/

6.1.2 eSignet github

GitHub - mosip/esignet at release-1.2.x

6.1.3 eSignet Mock Implementations Repo

GitHub - mosip/esignet-mock-services: Repository contains mock implementation of auth for e-signet

6.1.4 Mock Relying Party Application

Back end

https://github.com/mosip/esignet-mock-services/tree/master/mock-relying-party-service Connect your Github account

Web User Interface

https://github.com/mosip/esignet-mock-services/tree/master/mock-relying-party-ui Connect your Github account

6.2 eSIGNET API Reference

6.2.1 MOSIP Stoplight

e-Signet | Identity Provider

6.2.2 eSignet Wallet Binding API

Wallet Binding Endpoint | Identity Provider

New API to be created here

https://github.com/mosip/esignet/blob/master/esignet-service/src/main/java/io/mosip/esignet/controllers/KeyBindingController.java

https://github.com/mosip/esignet/blob/master/binding-service-impl/src/main/java/io/mosip/esignet/services/KeyBindingServiceImpl.java

https://github.com/mosip/esignet/blob/master/esignet-core/src/main/java/io/mosip/esignet/core/spi/KeyBindingService.java

reference Application

Health Services

6.3 WebAuthn References

Web Authentication: An API for accessing Public Key Credentials - Level 3