SUV Design

Version	Date	Author	Comment
1.0	05 July 21	David Chadwick and Ioram Sette	Initial release

Requirements

1. Each wallet SHOULD create a VP of a type specific to its wallet e.g. WalletXXX - see Appendix 1. If it does not, then the SUV will attempt to determine the correct verifier based on the DID method and JSON-LD proof type.
2. Each VC SHOULD contain a Terms of Use saying what trust framework it is part of, see Appendix 2. If it does not, or if the TRAIN API fails, then the SUV will have a configured list of trusted Issuers which is called by default.
3. Each Verifier SHOULD support the simple verifier API that passes in the VP and returns the validated VCs along with the challenge, see Appendix 3. If it does not we will see about including Verifier specific APIs in the SUV database.
4. The SUV is called via the Access Decision API, see Appendix 4.

SUV Algorithm

1. Receive a set of VPs from the RP in OIDC format (see Appendix 1) via the Access Decision API (see Appendix4)
2. Loop over the set of VPs. For each VP:
    1. If it is a jwt_vp decode the JWT.
    2. Obtain the type of VP, the DID method of the VP Holder, and the cryptographic algorithm used for its signature.
    3. Call the Verifier API Chooser procedure (see Appendix 8)
    4. Call the Verifier that supports this VP type using the Verifier API (see Appendix 3). The SUV will contain details of the VP types that are supported and the Verifiers to call for each one. Note. For Phase 2 it MAY be possible to use different APIs for different Verifiers if the SUV contains the details of the different APIs.
    5. Get back all the validated VCs less the crypto (in standard W3C JSON format).
    6. Note. In the current design the Challenge checking is performed by the Verifier API and not by the SUV).
3. Loop over the set of verified VPs. For each VP:
    1. Loop over the set of VCs. For each returned VC:
        1. Extract a list of trust schemes from the VC's terms of use and pass each of these along with the VC Issuer to the Train API, see https://app.swaggerhub.com/apis/train8/atv/1.0.0#/TRAIN_ATV_Request_Params. If there is no TRAIN ToU present in the VC, or if the TRAIN API says the Issuer is not trusted, then call the default SUV trusted issuers list to see if the Issuer is included in the Trusted Issuers table.
        2. If the VC Issuer is trusted keep the VC else throw away the VC and return an error (003 one or more of the VC Issuer's are not trusted).
        3. For phase 2 (ignore in phase 1): Check the type of the VC and retrieve the schemas matching the type stored in the SUV Database. Check that the VC properties conform to the schemas for the type.
4. For Phase 1.Call the PolicyMatching API (see Appendix 6) passing it the validated VCs and the Policy Ref and Policy Match, and use proprietary code to match the VCs to the policy.
5. For Phase 2. Retrieve the policy from the policy server (see Appendix 5) and write open source policy matching code.
6. Assuming the policy matches, if the config option requires it, map the validated VC properties into OIDC claims and return them to the RP, else return the validated VCs.

Appendix 1. VP format from current OIDC draft “OpenID Connect for Verifiable Presentations”


```
[
   {
      "format":"jwt_vp",
      "presentation":
“eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImRpZDpleGFtcGxlOmFiZmUxM2Y3MTIxMjA0MzFjMjc2ZTEyZWNhYiNrZXlzLTEifQ.eyJzdWIiOiJkaWQ6ZXhhbXBsZTplYmZlYjFmNzEyZWJjNmYxYzI3NmUxMmVjMjEiLCJqdGkiOiJodHRwOi8vZXhhbXBsZS5lZHUvY3JlZGVudGlhbHMvMzczMiIsImlzcyI6Imh0dHBzOi8vZXhhbXBsZS5jb20va2V5cy9mb28uandrIiwibmJmIjoxNTQxNDkzNzI0LCJpYXQiO…..8PHbF2HaWodQIoOBxxT-4WNqAxft7ET6lkH-4S6Ux3rSGAmczMohEEf8eCeN-AfQGbg"
   },
   {
      "format":"ldp_vp",
      "presentation":{
         "@context":[
            "https://www.w3.org/2018/credentials/v1"
         ],
         "type":[
            “VerifiablePresentation",“WalletXXX"
         ],
         "verifiableCredential":[
            {
               "@context":[
                  "https://www.w3.org/2018/credentials/v1",
                  "https://www.w3.org/2018/credentials/examples/v1"
               ],
               "id":"https://example.com/credentials/1872",
               "type":[
                  "VerifiableCredential",
                  "IDCardCredential"
               ],
               "issuer":{
                  "id":"did:example:issuer"
               },
               "issuanceDate":"2010-01-01T19:23:24Z",
               "credentialSubject":{
                  "given_name":"Fredrik",
                  "family_name":"Strömberg",
                  "birthdate":"1949-01-22"
               },
               "proof":{
                  "type":"Ed25519Signature2018",
                  "created":"2021-03-19T15:30:15Z",
"jws":"eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..PT8yCqVjj5ZHD0W36zsBQ47oc3El07WGPWaLUuBTOT48IgKI5HDoiFUt9idChT_Zh5s8cF_2cSRWELuD8JQdBw",
```

```
                    "proofPurpose":"assertionMethod",
                    "verificationMethod":"did:example:issuer#keys-1"
                }
            }
        ],
        "id":"ebc6f1c2",
        "holder":"did:example:holder",
        "proof":{
            "type":"Ed25519Signature2018",
            "created":"2021-03-19T15:30:15Z",
            "challenge":"()&)()0__sdf",
"jws":"eyJhbGciOiJFZERTQSIsImI2NCI6ZmFsc2UsImNyaXQiOlsiYjY0Il19..G
F5Z6TamgNE8QjE3RbiDOj3n_t25_1K7NVWMUASe_OEzQV63GaKdu235MCS3hIYvepc
NdQ_ZOKpGNCf0vIAoDA",
            "proofPurpose":"authentication",
            "verificationMethod":"did:example:holder#key-1"
        }
    }
]
```


Appendix 2. TRAIN Terms of Use

Each VC must contain a Terms of Use contains its trust framework

```
"termsOfUse": [{
    "type": "https://train.trust-scheme.de/info/",
    "trustScheme": ["<member of trustscheme1>", "<member of trustscheme2>"]
  }]
```

where each member of trust scheme is a DNS name.

Appendix 3. Verifier API

There are a number of different options available to this API call. Either the Verifier could be passed the challenge and asked to check that it is correct, or the Verifier could return the challenge and the SUV check it. Either the Verifier could check that the VC Issuers are trusted, or the Verifier can return the Issuers and the SUV can perform the check.
In the current version we have decided to make the Verifier as simple as possible and put the work inside the SUV

Input Parameters

VP: A single VP in either JWT or JSON-LDP proof syntax

Return Codes

200 (OK) along with the validated VCs with their proofs removed, in the W3C format

The following error codes are returned:
400 (Bad Request). The VP or an embedded VC is badly formatted or the cryptography failed
401 (Unauthorised) VC not issued to VC Holder or VCs not all issued to VC holder or VC delegation problem or mismatch between VP and embedded VCs
403 (Forbidden) The VP was not issued to the RP.
404 (Not found) The endpoint was not found on the server.

406 (Not Acceptable) VP or VCs are expired or signature verification fails or VP already verified (replay attack detection).
415 (Unsupported Media Type) incorrect/unknown schema elements in the VP or a VC.
417 (Expectation Failed) the challenge in the VP does not match the input challenge.
500 (Internal Server Error) - something went wrong with the Verifier

Appendix 4. The Access Decision API

Input Parameters

VPs: the list of VPs to be validated (see Appendix 1)
RP-URL: the URL that the VPs are targeted at (i.e. it should be in the audience field of the signed VPs)
Challenge: a random string that should be present in the signed VPs
PolicyRegistryURL: the URL of the public policy registry to be contacted
PolicyMatch: A JSON object comprising an action and a target, or a type (could be concatenated to the PolicyRegistryURL),

---

Return Codes

200 OK along with Granted (granted=true) and the set of validated attributes (in either OIDC format or VC format) is returned if the VP and embedded VCs are cryptographically correct, the audience field matches, the VCs belong to the holder, the Issuers are trusted and the VCs match the RP's policy

200 OK along with Denied (granted=false) and a Reason Code is returned when the VP and embedded VCs are cryptographically correct. The following reason codes are defined: 001 the VP or the VCs do not belong to the holder (Proof of possession fails); 002 the VCs do not match the RP's policy; 003 one or more of the VC Issuer's are not trusted; 004 the challenge in the VP does not match the input challenge; 005 incorrect/unknown schema elements in the VP or a VC; 006 the VP or the VCs are expired or revoked.

The following error codes are returned:
400 (Bad Request). The VP is badly formatted, or the cryptography fails, or a request parameter is missing or badly formed
403 (Forbidden). The VP audience does not match the RP URL
404 (Not found). The policy cannot be found in the policy registry, or a Verifier cannot be found to match the VP type
500 (Internal Server Error) - something went wrong with the SUV e.g. database error or other bug
503 (unavailable) - if a Verifier or Policy Registry or Internal DB are currently not available.

Appendix 5. The Policy Registry API

Input parameters

PolicyMatch: A JSON object comprising an action and a target, or a type
SupportedTypes: a list of strings, each string being one supported policy types/syntax. e.g. DIF_PE, or Identiproof

---

Return Codes

200 OK along with the required policy type in the required syntax

The following error codes are returned
400 (Bad request). The request was badly formatted
404 (Not Found). The policyMatch was not found in the policy registry
415 (Unsupported Media Type) The Supported Types is not supported by the Policy Registry

Appendix 6. The internal Policy Matching API

Input parameters

PolicyRegistryURL: the URL of the public policy registry
PolicyMatch: A JSON object comprising an action and a target, or a type
VCs: The set of trusted VCs with their proofs removed

Return Codes

200 (OK). Along with Matched = True or False. Along with False the following reason codes are defined: 001 Insufficient VCs to match the policy

The following error codes are returned:
400 (Bad Request). A request parameter is missing or badly formed
404 (Not found). The policy cannot be found in the policy registry
500 (Internal Server Error) - something went wrong with the Policy Matcher e.g. database error or other bug such as Policy is badly formatted
503 (unavailable) - The Policy Registry is currently not available.

Appendix 7. Mapping to OIDC Claims

Input: A verified credential with its proof removed
Output: A list of OIDC claims to be returned to the RP

There will be a configuration table comprising 2 columns: OIDC claim name (which might be a name registered with IANA or a local claim name), VC property URI.

The mapping procedure will go through the list of verified subject properties in the VC, converting the VC local property name into its full URI by using the @context value, then looking up the URI in the table and returning the OIDC claim name.

The rationale for this procedure is that two different VCs could have the same property name which is associated to different @contexts and has different URIs. Thus they need to be mapped into different OIDC claim names.

There will be a configuration file containing all the @context definitions that the SUV knows about (i.e. URI and user friendly property names). Each VC may have multiple @contexts with subject properties from all of them. The routine will pick up the first @context from the input VC, then move through the subject property names looking for a match. When a match is found the local property name is replace with the URI. Once the subject object has been processed, the next @context is picked and the process repeated. Once all @contexts have been processed a check is done to ensure that each subject property has been converted into its URI value. If not, an error is returned “unrecognised subject property <name>”
Once all the subject properties have been converted into their URIs, the properties are then turned into OIDC claims and returned to the caller.
If there are VC properties that do not have an entry in the OIDC claims table, then the VC property name is returned as a URL.

Appendix 8. Determining which Verifier to call (Chooser)

Determining the correct Verifier routine to call is based on several factors:
- the VP type. Some wallets produce product specific VP types

- the DID method. Some wallets produce issuer and holder DIDs with specific DID methods
- the crypto algorithm. Some wallets produce LD Proofs using one type of cryptographic algorithm.

The input parameters are: the VP type (could be missing), the DID method, the cryptographic algorithm being used.
The output parameter are details of the Verifier API to be called (i.e. a URL in phase 1)
The procedure is configured with a table comprising 4 columns: the VP type, the DID method, the cryptographic algorithm and the URL of the Verifier API to be called. Each of the table entries may comprise: a string value, * (meaning any), and - (meaning null).

The procedure takes the input arguments and looks through the table for a match. If no match is found then an error is returned (“No configured Verifier”), otherwise the matched Verifier URL is returned.

Note. For testing purposes there should be a final row containing *, *, * (i.e. match all) and it should be configured with the URL of the NoCryptoCheck Verifier API, which will take in the VP, remove all the proofs, throw them away and return the validated VCs.