Protocol: web2app
Version: 2.0
Document Version: 2.6
Category: Experimental

Farid Ismayilzada
SIMA
AzInTelecom LLC

Web2app
2.0
IDP Integration Protocol

Status of this Memo

This memo defines an Experimental Protocol for the community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Abstract

The Web2app IDP Integration Protocol serves as a comprehensive guide detailing the seamless integration process between an Identity Provider (IDP) and a Service Provider. This protocol has been meticulously crafted, focusing primarily on catering to the unique requirements of mobile device-based keyholder applications. This protocol documentation delves into the technical nuances, providing clear and concise explanations accompanied by illustrative examples. By adhering to the guidelines outlined herein, developers and system architects can implement this integration protocol effectively, fostering a secure environment for users of mobile keyholder applications.

Document versioning

Version	Date	Comment	Author
1.3	31.03.2022	Initial Version of web2app	Farid Ismayilzada
1.4	21.03.2022	- Update of protocol version: **web2app 1.1** - Added **ClientName** to the field **ClientInfo** - Added **DataURI** to the field **DataInfo** **Check the references : 4.1 , 4.3 , 5.1**	Farid Ismayilzada
1.5	30.07.2022	- Added RedirectUri field: SignableContainer.ClientInfo.**RedirectURI in protocol version 1.3**	Farid Ismayilzada
2.5	10.10.2023	- Concept changes - Proof of Identity added - Proof of actions added - Update data exchange model between IDP and RP	Farid Ismayilzada

Content

1.Introduction

1.1 Scope

This document explains a secure data exchange protocol called "web2app." In this document, there are two main roles: the provider and the consumer. Additionally, there are two parties involved: the service provider and the identity provider (refer to section 1.2).

1.2 Abbreviations

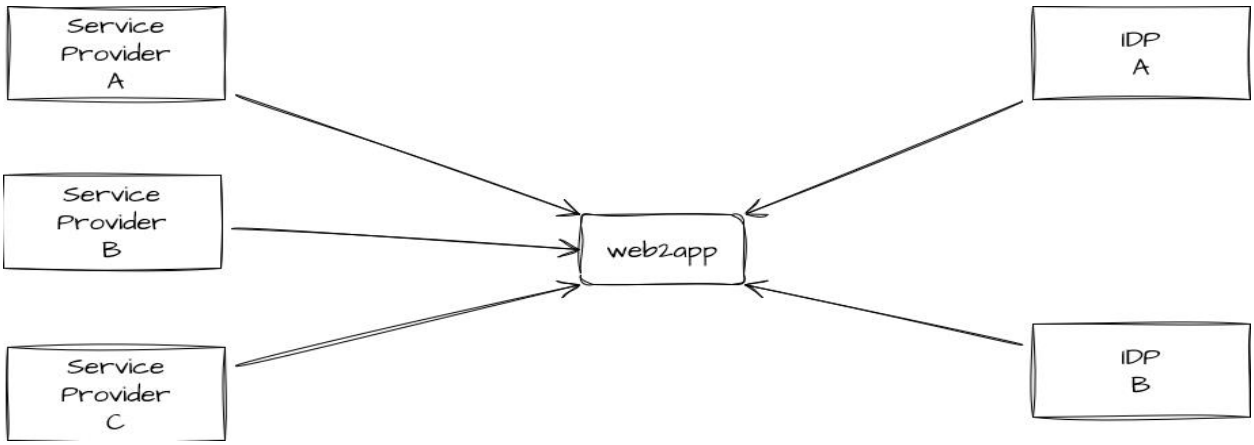
Abb	Name	Description
SP	Service Provider, Resource Provider	The client will integrate into the identity provider
IDP	Identity Provider	The system which provides trusted identity
TSA	Time Stamp Authority	
CA	Certificate Authority	
H()	Hash function or checksum algorithm	SHS256 , SHA1 ,etc
CH	Checksum	

2. Information

2.1 Initials

To utilize the protocol, the client must possess the following initial parameters, which will be supplied by the identity providers:

- **Client Id**: The identifier provided by the identity provider.
- **MasterKey**: Shared key that is shared between IDP and service provider.
- **TheKeyAlgName**: The algorithm name approved by the identity provider.s
- **Trents**: List of trusted roots that the IDP provides. At least one root should be provided.
- **ChecksumAlgorithm :** Defined hash algorithm for preparing the checksum of the contract

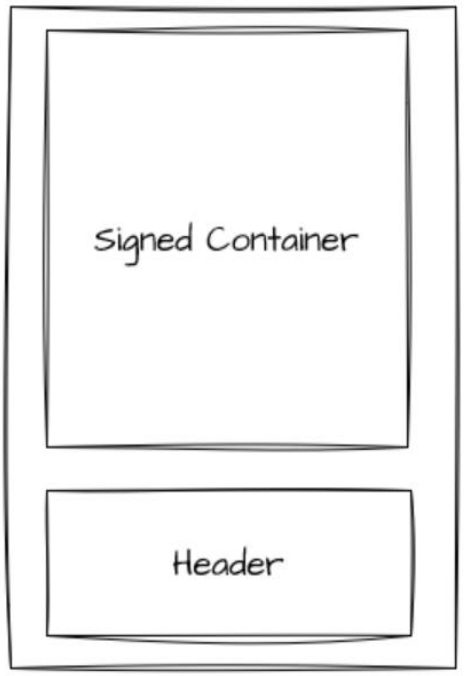


The Service provider does not have direct integration with IDP. The data exchange happens only when both sides agree to negotiations.

3. Contract Generation

3.1 Contract Details

Contract is the main element for negotiation between IDP & Service Provider.



This is the structure of tscontract. The encoding format of the contract is “json”.
Signed Container is the payload of the contract where the header is the information about the payload and contains validation mechanism. The contract is the projection of the data behind the scenes for each service provider. Single contract can represent single type-defined operation as well as multiple operations. For example, there might be the case that the client uses single contract with long life cycle that IDP users can sign the same contract multiple times and send to the same callback. The process can be used for voting, document signing, authentication etc..

#table 1.

Signable Container			
Name		**Description**	**Value**
ProtoInfo	**Name**	Protocol name.	**web2app**
	Version	The version of the protocol.	Version 2.0
OperationInfo	**Type**	Type of the operation. The reason the contract is generated.	Auth Sign
	OperationId	Transaction ID for the contract. Any string data.	Value: String
	NbfUTC	Not before or activation date as UNIX UTC timestamp.	INT Value Example: 1649331930
	ExpUTC	Not after or expiration date as UNIX UTC timestamp.	INT Value Example: 1649331930
	Assignee	User Filter for the client. It works with different options. If The assignee exists, IDP will check the authorization of the filter	For example: “o_*”, “t_***” , ”s_***” , etc
DataInfo	**DataURI**	GetFile(getdata) URI.	URL where which is responsible for providing data
	AlgName (Optional)	The name of Hash Algorithm	SHA256, SHA1, RIPEMD160, RIPEMD256
	FingerPrint (Optional)	Checksum of the data behind. base64 format	base64encodeded(#checksum)
ClientInfo	**ClientId**	Client ID that is provided by IDP	Int Value: Not null
	ClientName	The name of the resource Application	Name of the client that will be displayed on IDP
	IconURI	Public Icon URI	Link to the icon (png,jpg,svg..)
	Callback	Public callback URL of the system	Example: https://scanme.sima.az/Home/callback
	RedirectURI Optional	Redirect UrI for app to redirect	Post sign redirect action

#table 2

Header		
Name	**Description**	**Value**
AlgName	Signature Algorithm name for the contract. Mainly it uses MAC with hash algorithm. By default, the checksum algorithm is optional. And if it was not defined in the algname the default value will be SHA256 for checksum.	Value string Enum name: HMACSHA256, SHA256RSA With defined Checksum algorithm it will be as follows: Full name of the algorithm “HMACSHA256” will be. HMACSHA256 = SHA256_HMACSHA256 If you want to change checksum algorithm to SHA1 it will be SHA1_HMACSHA256 Checksum algorithm can be selected from list of supported hash algorithms.
Signature	Digital Signature of the contract	Base64 encoded signature. H() - hash function (by default SHA256) K - Secret key (master Key or Shared Key) S - Signature CH - checksum with H() Signature creation process: Algorithm = CH_SAlgName CH=H(SignableContainer) S=HMAC(CH , K) => EncodeBase64(S) S=SHA256RSA(CH,K) => EncodeBase64(S)

Map of the Signature Algorithms
#table 3

Sign Alg	Checksum			
	SHA	RIPEMD	GOST3411	Blake3
HMACSHA256	SHA1_HMACSHA256, SHA256_HMACSHA256 (**HMACSHA256**), SHA384_HMACSHA256, SHA512_HMACSHA256	RIPEMD128_HMACSHA256 RIPEMD160_HMACSHA256 RIPEMD320_HMACSHA256	GOST34112012256_HMACSHA256 GOST34112012512_HMACSHA256	Blake3_HMACSHA256
HMACSHA384	SHA1_HMACSHA384, SHA256_HMACSHA384 (**HMACSHA384**), SHA384_HMACSHA384, SHA512_HMACSHA384	RIPEMD128_HMACSHA384 RIPEMD160_HMACSHA384 RIPEMD320_HMACSHA384	GOST34112012256_HMACSHA384 GOST34112012512_HMACSHA384	Blake3_HMACSHA384
SHA256RSA	SHA1_SHA256RSA, SHA256_SHA256RSA (**SHA256RSA**), SHA384_SHA256RSA, SHA512_SHA256RSA	RIPEMD128_SHA256RSA, RIPEMD160_SHA256RSA RIPEMD320_SHA256RSA	GOST34112012256_SHA256RSA GOST34112012512_SHA256RSA	Blake3_SHA256RSA
SHA384RSA	SHA1_SHA384RSA, SHA256_SHA384RSA (**SHA384RSA**), SHA384_SHA384RSA, SHA512_SHA384RSA	RIPEMD128_SHA384RSA, RIPEMD160_SHA384RSA RIPEMD320_SHA384RSA	GOST34112012256_SHA384RSA GOST34112012512_SHA384RSA	Blake3_SHA384RSA

If the checksum algorithm is not defined in the contract, the default algorithm for the checksum is “**Sha256**”.

3.2 Contract Example

To generate the contract, you'll require both the client ID and its associated key, commonly referred to as the master key. The nature of the contract depends on the specified type. When the type is Auth, the contract may consist of random challenge data (a random nonce) or small data preferred by the resource service. On the other hand, if the type is Sign, the contract should encompass the document, such as a PDF file or signature related data.

The Resource Service organizes data based on individual contracts. This service is designed to integrate with an Identity Provider and can be utilized in various web-related systems without any limitations. It allows data to be associated with multiple contracts, and conversely, a single contract can be linked to multiple sets of data.

There are two types of contracts depending on “**OperationInfo.Type**”:

- Auth - Authentication Contract
- Sign - Signature Contract

Contract specification for the auth and sign is the same. Difference lies in the next stage aspects.

```
{
      "SignableContainer": {
                "ProtoInfo": {
                          "Name": "web2app",
                          "Version": "2.0"
                },
                "OperationInfo": {
                          "Type": "Auth",
                          "OperationId": "123456789",
                          "NbfUTC": 1649721600,
                          "ExpUTC": 1650326400,
                          "Assignee": [

                          ]
                },
                "DataInfo": {
                          "DataURI": "https://scanme.sima.az/home/getdata"
                },
                "ClientInfo": {
                          "ClientId": 1,
                          "ClientName": "ScanMe APP",
                          "IconURI": "https://aaaaaa.png",//Icon Pulic URL
                          "Callback": "callbackURL",
                          "RedirectURI": "test.az/tran/123456789" // Post Sign action
                }
      },
      "Header": {
                "AlgName": "HMACSHA256",
                "Signature": "nQxNMasxoL1WuLJ2x1kRhFAmwFTbDxyjMGnF1Tycyr0="
      }
}
```

One of the main differences in this version is that **DataInfo.DataURI** is required here. Service Provider should put **GetFile** Endpoint here.

According to operation type, **sign** is responsible to provide the data that should be signed in IDP side. It can be document (PDF) or hash of the document that should be signed by IDP.
For the **auth** operation type there is no strict requirement that what kind of data should be returned. It is up to service provider what to return there.

Additionally, “Assignee” field also developed with default and additional filters. It means that, IDP (Identity Provider) will provide supported available filters for itself, and SP (Service Provider) can use those filters in contract generation. Default supported filters are:

- t_ - it defines the client type of the IDP. For example: t_a it means the app of the IDP. t_s means the SDK of the IDP can apply for this filter. t_* are not allowed.
- p_* - the identity number filter. If this filter exists in a contract, it means that only personal certificates should be applied for this contract. If the contract is generated for the specific person with specific id, * must be replaced with that id.
- o_* - It defines the organization identity number. On most cases it is tax Id number. If contract is generated with that filter only organization certificates can be applied.

There are also opposite filters as follows:

- t!_ - the opposite filter of t_. For example, t!_a means, the application type of the IDP is not applying this filter. t!_* are not allowed.
- p!_* - the opposite filter of p_*. Prohibition of personal certificates. Specific certificates can be forbidden with this filter as: p!_XXXXXXX.
- o!_* - the opposite filter of o_*. Prohibition of organizational certificates. Specific certificates can be forbidden with this filter as: o!_XXXXXXX.

NOTE: ***Duplications, usage of generic opposite filters and opposite filters with same value are not allowed in the filters. The filters should be unique. Example in all combinations*:**

```
"Assignee": [
          "p_EXPIN", "p!_EXPIN",
          "p_*", "p!_*",
          "t_s", "t!_s",
          "t_*",
          "t!_*"
          "o_*", "o!_*",
          "o_EXTINCODE","o!_EXTINCODE"
]
```

3.3 Contract Presentation

After contract is built, it should be generated to proper format to be recognized by IDP applications. Proper format is considered as QR code or link. For presenting the contract you need to initiate some parameters.

- tsquery – the encoded contract itself (base64 format)
- tscta – stands for **compression type algorithm** for contract.

Contract generation Steps:

1. Generation
2. Signing
3. Encoding (compression if exists)
4. Presentation

#generation

The generation of the contract means all required parameters in contract must be filled and serialized.

Q = Serialize(payload)

Q – becomes a serialized contract payload

#signing

The Q data should be signed by SP private key and prepared as a Signed Contract.

S= Sign (k,Q,a)

k – key

a – sign algorithm

S – signature of the payload

Then take the Q and S next prepare signed contract model.

SC = build(Q,S)

#encoding

In this stage compression also can be applied to the contract. But it is optional.

EncodedTSC = Encodebbase64 (Compress(alg , SC))

alg - is compression algorithm name

Supported algorithms: **gzip ,deflate, br**

#presentation

Contract contained parameter **tsquery** is defined to keep the encoded contract in the presentation URL. **tsquery** can be pinned either to the web link or any protocol undefined URI. Example :

{idp}://web2app?tsquery={base64encoded-tsquery}
{idp}://web2app?tsquery={base64encoded-tsquery}&tscta=gzip
https://web2app.az/contract?tsquery={encodedContract}
https://web2app.az/contract?tsquery={encodedContract}&tscta=gzip

A QR code and deeplink are supported for all these variations.

4. Contract Usage

4.1 GETDATA

This function should be implemented by the Resource Provider. It allows us to get the data behind the generated contract. It works through the HTTP protocol. The Identity provider will always call this function with authorization parameters. There are several ways of providing the data behind the contract. The service provider can provide the data with or without authentication, but in any scenario, the identity provider will always provide the identity of the end user. In other words, if it is without authentication, the service provider will bypass the authentication process. (signature verification, etc).
After parsing the qr contract, the identity provider calls the **GETDATA**() function.

One of the major parts that needs to be mentioned is GETDATA() is included inside the contract as **DataURI**.

GETDATA response model:
Content-Type: application/json

#table 4

		Getdata response model	
Name		Value	Description
sessionid	String		**Sessionid defined by service provider**
type	url, raw		**Property is optional. If not exists, default = raw** url – link to download the data. it can be the link from object storage. raw – base64 raw data
dataObjects[]	-name	String	String name for contract data
	-data	String	If the **type = url** For Example, data= https://web2app-api.sima.az/api/v1/getDocument?token=58935180f7dfa9d4c4fe82214cf5a58a if **type = raw** data= base64 string of the data object
signFormat	String: **-pades-b,** **-pades-t,** **-hash,** **-hash_{algname}**		**Sign contract - Property is optional. If not exists, default = pades-t for signature contracts.** **Authentication contract - this param unnecessary.**
claims[]	String [] Values: **-deviceinfo,** **-location,** **-idinfo,** **-idphoto,** **-certcollectioninfo** **-validate** **-validateddata**		**deviceinfo** – device information: model, Id , etc **location** – location info provided by IDP user: latitude, longitude. **idinfo –** Id information of the user **idphoto –** photo from ID document **certcollectioninfo –** list of all certificates inside IDP APP **validate -** function that validate identity using supported mechanisms inside IDP such as biometry validation. **validatedata –** validate and return validation evidence to the service provider
message	String		Message from the SP. It will be used when for some reason the data will not be provided. Error messages will be shown on IDP screen.

Claims – the data that will be asked from IDP (Identity provider) by SP (Service Provider). Those data are flexible and can be changed shrined or expanded.

Default Claims:

1. Device Information that IDP can offer.

```
{
    "name": "device name",
    "osversion": "osversion",
    "appversion": "IDP app version",
    "apptype": "IDP app type"
}
```

2. Location information

```
{
    "latitude": "latitude of the mobile user",
    "longitude": "longitude of the user "
}
```

3. IdPhoto - string base64 photo of the ID card

4. Validate – is the function that can be claimed from SP and IDP should apply it. It is second factor of the authentication before using the digital signature keys. For example, it this parameter exists the IDP, can validate the identity of the user using biometric verification services.

5. Validated Data – this is evidence of the previous function that applied by IDP. In other words, the result of the validate() function should be sent to the SP. Structure is as follows:

```
{
    "dataid": "evidence id of the validation",
    "data": "base64 result of the validataion "
}
```

6. certcollectioninfo - certcollectioninfo information from IDP . All certificate inside the IDP app can be requested by SP

```
{
    "certificates": [
        {
            "certSerialNumber": "certificate serial number",
            "type": "Personal/Commercial",
            "rawCertificate": "base64 certificate"
        }
    ]
}
```

7. Id info

```
{
    "header": {
        "alg": "SHA-256withRSA",
        "pattern":
"PersonId/RequestId/Name/Surname/Middlename/DocumentNumber/CountryCode/CreateTimeUTC/
BioTransactionId/OptionalData.DocumentPin",
        "signature":
"dkBc9nY8JW5P8wCyZVkVmHqEhu8FDy9LjNgxzqUqe8hZiLxMEBMHhjdYCDHqrqj61Z8YkrHWz5anpC4WP4Q1
eEBpysxwnNEpn39Mv3qn7RBfFppSigujEMHevifZdBGWNxjrvhSS6bsFrpF2zLoBaWtrcnoF7FELaVNFMmCTG
zK4W46Q5ubVqjGZjF6HwwuEqmfGkHSRPwDftu1nxBL1AL91df8Aza4Yx1pVDKRDy3LAybMTiZ5DATZ9A6qRzk
RCMZr2EcnmSXqWx7dCvFfTXG72BxojLZvjbes5oaDWi7yaNnrfWhGDucTTZVns7od9KAQgMbmC4ikRLNyQChi
Nr91pVein7"
    },
    "payload": {
        "bioMetrics": {
            "bioMetricsInfo": {},
            "bioMetricsType": "SVT"
        },
        "bioTransactionId": "296689ae-ece9-413f-ae6d-9ba61dXXXXXX",
        "countryCode": "AZE",
        "createTimeUTC": "2024-04-03T08:53:09.5901592Z",
        "docType": "ID",
        "docIssuedDate": "issued date ",
        "docExpiredDate": "doc expired date",
        "sex": "M",
        "birthDate": "date of birth",
        "documentNumber": "AA11222333",
        "middlename": "Father name",
        "name": "Farid",
        "optionalData": {
            "documentPin": "XXXXXXX"
        },
        "personId": "AZE_I_XXXXXXXX",
        "requestId": "LYo2LbA4lf/fm9tmlpy7Q2eJisQX9HK/t2ufQcFUJTg=",
        "surname": "Ismayilzada"
    },
    "version": "1.0.1"
}
```

Additionally, more custom data can be added to the claims, depending on the negotiation between IDP and SP.

4.2 Callback

Callback to SP – IDP will request SP after consuming the data from GETDATA endpoint of the SP. The main purpose of the callback is to deliver the signed data or authentication challenge, with additional requested claims if demanded. In both operations IDP will post back to the SP signed data that consumed from GETDATA. Callback endpoint is defined inside the TsContract (described in #table 1).

#table 5

Callback request model		
Name	Type	Description
sessionid	String	**sessionId consumed from getdata**
type	String auth/sign	**Type of the operation described in #table 1.**
operationId	String	**Operation Id inside the TS contract #table 1**
dataSignature	String	**Base64 signed data**
token	String optional	**IDP transaction token**
kid	String base64	**Key identifier for client** **K – shared key bytes** **S – Contract signature** **S= TsContract.Header.Signature.Bytes[]** **Alg – Checksum algorithm that is defined by IDP. IDP can use the algorithm that defined inside TsContract. But it is not mandatory to use exact the same one. It can be chosen among the supported ones.** **P=Concat(S,k)** **Kid = CheckSum(alg , P)**
signedDataHash	String base64	**Checksum (Data) – Optional**
signFormat	String: **-pades-b,** **-pades-t,** **-hash,** **-hash_{algname}**	**Sign contract - Property is optional. If not exists, default = pades-t for signature contracts.**
claims[]	type	Type – one of the types in #table 4 claims. Or custom string type between IDP and SP
	value	Dynamic json model or string
algName	String enum Optional	**SignedDataHash** checksum algorithm name **Optional if SignedDataHash doesn't exist**
dataName	String	**Name of the data that consumed from SP #table**
statuscode	Integer optional	**2XX , 4XX ,5XX** **Optional. IDP can post back to SP for any reason even the consuming process ended with failure.**
message	String optional	Optional. message to send to SP if error occurred or something else

Response Model

Callback request model		
Name	Type	Description
status	String	Status provided by Service Provider
message	String	Message provided by Service provider. IF there is an error on callback IDP will show the message

5. IDP Authentication by SP

In order to validate the authenticity of each request SP have to validate IDP requests. For IDP's, that applies this protocol, requests to SP need to be digitally signed. For this matter we are signing all http requests that come out IDP.

Headers		
Name	**Description**	**Value**
ts-sign-alg	String enum that defines the signature algorithm	ECDSA_SHA256 / RSA_SHA256 etc...
ts-cert	String - base64	The x509 certificate of the client
ts-sign	String - base64	Signature of the request If the http Method = POST Signature = sign(request.Body,ts-sign-alg , privKey) Method = GET Payload = DataURI path without domain sign(payload,ts-sign-alg , privKey)

How to validate authenticity?
IDP have to share or publish list of supported issuer certificates with SP.

When SP receives IDP requests, first:
Validate the integrity of the request itself.
Then, validate the certificate with issuer list.

6. Examples

Contract Examples:

1. Contract for Organizational Identities

```
{
  "SignableContainer": {
    "ProtoInfo": {
      "Name": "web2app",
      "Version": 2
    },
    "OperationInfo": {
      "Type": "Auth",
      "OperationId": “541616416”,
      "NbfUTC": 1712275200,
      "ExpUTC": 1713571200,
      "Assignee": ["o_*"]
    },
    "DataInfo": {
      "DataURI": "https://scanme.sima.az/home/getdata"
    },
    "ClientInfo": {
      "ClientId": 2007081,
      "IconURI": "https://play-
lh.googleusercontent.com/ShzfzS2goiwJtQOjHGoxqRUuOuSYGb-xuuMpe_jhqAFsRph9gRhYu3xb-
xpwoo8H59w",
      "Callback": "https://scanme.sima.az/home/callback",
      "ClientName": "Web2APP Client"
    }
  },
  "Header": {
    "AlgName": "SHA256_HMACSHA256",
    "Signature": "EOiKd2v4Ublyu/3wAMwBXDi/vjzijO1sdsxw6tbF8tU="
  }
}
```

IDP will request HTTP GET to DataURI :

```
HTTP GET https://scanme.sima.az/home/getdata
ts-cert : certificateBase64
ts-sign : signatureBase64
ts-sign-alg :
```

SP will validate the authenticity of the request with signature and certificate issuer.
After validation SP will produce the response. Example Response:

```
{
    "type": "url",
    "dataObjects": [
        {
            "name": "file1",
            "data": "http://fileurl.az/21155151515"
        }
    ],
    "claims": [
        "deviceinfo",
        "validate"
    ],
    "message": “”
}
```

SP wants from IDP to sign the document “file1” to be signed and additionally SP wants to be sure that the identity will be validated before sign action and device info will be sent back with signed document.

```
{
  "type": "sign",
  "operationId": "541616416",
  "sessionId": "123123123541616416",
  "dataSignature": "base64signed Document",
  "token": "rvtok_XXXXXXXXXXXXXX",
  "kid": "key identifier for client",
  "signFormat": "pades-t",
  "claims": [
    {
      "type": "deviceinfo",
      "value": {
        "name": "device name (Iphone 13)",
        "OSversion": "os version",
        "appversion": "3.1.1",
        "apptype": "a"
      }
    },
    {
       "type": "location",
       "value": {
             "latitude": "latitude of the mobile user",
             "longitude": "longitude of the user "
             }
    }
  ],
  "dataName": "file1",
  "statuscode": "200",
  "message": "Signed"
}
```

Response example from service provider

```
{
    "status": "completed",

    "message": “Data uploaded Successfully”
}
```

7. Conclusion

In wrapping up, the "Web2app 2.0 IDP Integration Protocol" provides a robust framework for facilitating seamless integration between Identity Providers (IDPs) and Service Providers (SPs), with a specific focus on mobile keyholder applications. This protocol document has been meticulously crafted to address the technical intricacies involved in data exchange, contract generation, and authentication processes.

We encourage developers and system architects to implement the guidelines outlined in this document to foster a secure environment for users of mobile keyholder applications.