注册过程

笔记本：	FIDO		
创建时间：	2015/10/27 16:58	**更新时间：**	2015/10/28 20:34
作者：	yangzhou1989@gmail.com		
URL：	https://fidoalliance.org/specs/fido-uaf-v1.0-ps-20141208/fido-uaf-protocol-v1.0-ps-20141208.html#widl-OperationHeader-serv...		

UAF Registration Request:

```
dictionary RegistrationRequest {
    required OperationHeader header;
    required ServerChallenge challenge;
    required DOMString       username;
    required Policy          policy;
};
```

例子如下：


```
[{
  "header": {
    "upv": {
      "major": 1,
      "minor": 0
    },
    "op": "Reg",
 "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
    "serverData": "IjycjPZYiWMaQ1tKLrJROiXQHmYG0tSSYGjP5mgjsDaM17RQgq0
dl3NNDDTx9d-aSR_6hGgclrU2F2Yj-12S67v5VmQHj4eWVseLulHdpk2v_hHtKSvv_DFqL4n
2IiUY6XZWVbOnvg"
  },
  "challenge": "H9iW9yA9aAXF_lelQoi_DhUk514Ad8Tqv0zCnCqKDpo",
  "username": "apa",
  "policy": {
    "accepted": [
      [
        {
          "userVerification": 512,
          "keyProtection": 1,
          "tcDisplay": 1,
          "authenticationAlgorithms": [
            1
          ],
          "assertionSchemes": [
            "UAFV1TLV"
          ]
        }
      ],
      [
        {
          "userVerification": 4,
          "keyProtection": 1,
          "tcDisplay": 1,
          "authenticationAlgorithms": [
            1
          ],
          "assertionSchemes": [
            "UAFV1TLV"
          ]
        }
      ],
      [
        {
          "userVerification": 4,
          "keyProtection": 1,
          "tcDisplay": 1,
          "authenticationAlgorithms": [
            2
          ]
        }
      ],
      [
        {
          "userVerification": 2,
          "keyProtection": 4,
          "tcDisplay": 1,
          "authenticationAlgorithms": [
```

```
        2
      ]
    }
  ],
  [
    {
      "userVerification": 4,
      "keyProtection": 2,
      "tcDisplay": 1,
      "authenticationAlgorithms": [
        1,
        3
      ]
    }
  ],
  [
    {
      "userVerification": 2,
      "keyProtection": 2,
      "authenticationAlgorithms": [
        2
      ]
    }
  ],
  [
    {
      "userVerification": 32,
      "keyProtection": 2,
      "assertionSchemes": [
        "UAFV1TLV"
      ]
    },
    {
      "userVerification": 2,
      "authenticationAlgorithms": [
        1,
        3
      ],
      "assertionSchemes": [
        "UAFV1TLV"
      ]
    },
    {
      "userVerification": 2,
      "authenticationAlgorithms": [
        1,
        3
      ],
      "assertionSchemes": [
        "UAFV1TLV"
      ]
    },
    {
      "userVerification": 4,
      "keyProtection": 1,
      "authenticationAlgorithms": [
        1,
        3
      ],
      "assertionSchemes": [
        "UAFV1TLV"
      ]
    }
  ]
],
"disallowed": [
  {
    "userVerification": 512,
    "keyProtection": 16,
    "assertionSchemes": [
      "UAFV1TLV"
    ]
  },
```

```
      {
        "userVerification": 256,
        "keyProtection": 16
      },
      {
        "aaid": [
          "ABCD#ABCD"
        ],
        "keyIDs": [
          "RfY_RDhsf4z5PCOhnZExMeVloZZmK0hxaSi10tkY_c4"
        ]
      }
    ]
  }
}]
```

关于header可见【名词解释】

下面是FIDO客户端发送请求给ASM的：
FIDO客户端解析注册请求的时候的步骤为：

- 选择主版本号是1，次版本号是0的消息m
- 解析消息m
  - 如果必要的UAF消息字段没有被设置或者不正确的被设置，则拒绝此次请求
- 显示可用的认证器给用户选择，其中必须去除被禁用的认证器
- 根据AppID，获取可信的应用的FacetID
  - 如果应用的FacetID没有在信任列表里，则拒绝此次操作
- 如果可用，请求TLS数据
- 计算FinalChallengeParams变量fcp，并且设置 *fcp.appID* , *fcp.challenge* , *fcp.facetID* 和 *fcp.tlsData*
  - 计算方法： FinalChallenge = base64url(serialize(utf8encode(fcp)))
- 对于所有符合UAF协议版本和用户同意的认证器：
  - 生成对应的ASMRequest数据
  - 将ASMRequest数据发送给ASM

1. Choose the message `m` with major version `1` and minor version `0`
2. Parse the message `m`
3. If a mandatory field in UAF message is not present or a field doesn't correspond to its type and value - reject the operation
4. Filter the available authenticators with the given policy and present the filtered authenticators to User. Make sure to not include already registered authenticators for this user specified in `RegRequest.policy.disallowed[].keyIDs`
5. Obtain `FacetID` of the requesting Application. If the `AppID` is missing or empty, set the `AppID` to the `FacetID`.

   Verify that the `FacetID` is authorized for the `AppID` according to the algorithms in [FIDOAppIDAndFacets].

   - If the `FacetID` of the requesting Application is not authorized, reject the operation
6. Obtain TLS data if it is available
7. Create a **`FinalChallengeParams`** structure `fcp` and set `fcp.appID`, `fcp.challenge`, `fcp.facetID`, and `fcp.channelBinding` appropriately. Serialize [RFC4627] fcp using UTF8 encoding and base64url encode it.
   - `FinalChallenge = base64url(serialize(utf8encode(fcp)))`
8. For each authenticator that matches UAF protocol version (see section Version Negotiation) and user agrees to register:
   1. Add `AppID`, `Username`, `FinalChallenge`, `AttestationType` and all other required fields to the ASMRequest [UAFASM].

      The FIDO UAF Client MUST follow the server policy and find the single preferred attestation type. A single attestation type MUST be provided to the ASM.

   2. Send ASMRequest to the ASM

上面的8.1步，形成【UAFASM】中的RegisterIn Object：

3.6.1 RegisterIn Object

WebIDL

```
dictionary RegisterIn {
    required DOMString      appID;
    required DOMString      username;
    required DOMString      finalChallenge;
    required unsigned short attestationType;
};
```

而形成的ASMRequest则包括：

All ASM requests are represented as ASMRequest objects.

WebIDL

```
dictionary ASMRequest {
    required Request requestType;
    Version asmVersion;
    unsigned short authenticatorIndex;
    object args;
    Extension[] exts;
};
```

3.3.1 Dictionary ASMRequest Members

requestType of type required Request
Request type

asmVersion of type Version
ASM message version to be used with this request. For the definition of the Version dictionary see [UAFProtocol]. The ASM version MUST be 1.0 (i.e. major version is 1 and minor version 0).

authenticatorIndex of type unsigned short
Refer to the GetInfo request for more details. Field authenticatorIndex MUST NOT be set for GetInfo request.

args of type object
Request-specific arguments. If set, this attribute MAY take one of the following types:

- RegisterIn
- AuthenticateIn
- DeregisterIn

exts of type array of Extension
List of UAF extensions. For the definition of the Extension dictionary see [UAFProtocol].

其中，requestType就包括GetInfo Request, Register, Authenticate, Deregister, GetRegistrations, OpenSettings
ebay源码中的request结构体：

```
package org.ebayopensource.fido.uaf.msg.asm;

public enum Request {
            GetInfo,
            Register,
            Authenticate,
            Deregister,
            GetRegistrations,
            OpenSettings
}
```

ASM向Authenticator发送和接受的命令都是TLV格式的，注册的Register Command(ASM向Authenticator发送)如下：

6.2.2.1 Command Structure

	TLV Structure	Description
1	UINT16 Tag	TAG_UAFV1_REGISTER_CMD
1.1	UINT16 Length	Command Length
1.2	UINT16 Tag	TAG_AUTHENTICATOR_INDEX
1.2.1	UINT16 Length	Length of AuthenticatorIndex (must be 0x0001)
1.2.2	UINT8 AuthenticatorIndex	Authenticator Index
1.3	UINT16 Tag	TAG_APPID (optional)
1.3.1	UINT16 Length	Length of AppID
1.3.2	UINT8[] AppID	AppID (max 512 bytes)
1.4	UINT16 Tag	TAG_FINAL_CHALLENGE
1.4.1	UINT16 Length	Final Challenge Length
1.4.2	UINT8[] FinalChallenge	Final Challenge provided by ASM (max 32 bytes)
1.5	UINT16 Tag	TAG_USERNAME
1.5.1	UINT16 Length	Length of Username
1.5.2	UINT8[] Username	Username provided by ASM (max 128 bytes)
1.6	UINT16 Tag	TAG_ATTESTATION_TYPE
1.6.1	UINT16 Length	Length of AttestationType
1.6.2	UINT16 AttestationType	Attestation Type to be used
1.7	UINT16 Tag	TAG_KEYHANDLE_ACCESS_TOKEN
1.7.1	UINT16 Length	Length of KHAccessToken
1.7.2	UINT8[] KHAccessToken	KHAccessToken provided by ASM (max 32 bytes)
1.8	UINT16 Tag	TAG_USERVERIFY_TOKEN (optional)
1.8.1	UINT16 Length	Length of VerificationToken
1.8.2	UINT8[] VerificationToken	User verification token

其中关于KHAccessToken见【名词解释】

Authenticator收到上面的Register Command后，authenticator做以下事情：

1. If this authenticator has a transaction confirmation display and is able to display AppID, then make sure `Command.TAG_APPID` is provided, and show its content on the display when verifying the user. Update `Command.KHAccessToken` with `TAG_APPID`:
    - Update Command.KHAccessToken by mixing it with Command.TAG_APPID. An example of such mixing function is a cryptographic hash function.

      > NOTE
      >
      > This method allows us to avoid storing the AppID separately in the RawKeyHandle.

        - For example: Command.KHAccessToken=hash(Command.KHAccessToken | Command.TAG_APPID)
2. If the user is already enrolled with this authenticator (via biometric enrollment, PIN setup or similar mechanism) - verify the user. If the verification has been already done in a previous command - make sure that Command.TAG_USERVERIFY_TOKEN is a valid token.
    1. If verification fails - return `UAF_CMD_STATUS_ACCESS_DENIED`
3. If the user is not enrolled with the authenticator then take the user through the enrollment process.
    1. If enrollment fails - return `UAF_CMD_STATUS_ACCESS_DENIED`
    2. If user explicitly cancels the operation - return `UAF_CMD_STATUS_USER_CANCELLED`
4. Make sure that Command.TAG_ATTESTATION_TYPE is supported. If not - return `UAF_CMD_STATUS_ATTESTATION_NOT_SUPPORTED`
5. Generate a new key pair (UAuth.pub/UAuth.priv)
6. Create a RawKeyHandle
    1. Add UAuth.priv to RawKeyHandle
    2. Add Command.KHAccessToken to RawKeyHandle
    3. If a first-factor authenticator, then add Command.Username to RawKeyHandle
7. Wrap RawKeyHandle with Wrap.sym key(这里wrap后就生成了加密后的KeyHandle，主要用于传给ASM)
8. Create TAG_UAFV1_KRD structure
    1. If this is a second-factor roaming authenticator - place key handle inside TAG_KEYID. Otherwise generate a random KeyID and place it inside TAG_KEYID.
    2. Copy all the mandatory fields (see section TAG_UAFV1_REG_ASSERTION)
9. Perform attestation on TAG_UAFV1_KRD based on provided Command.AttestationType.
10. Create TAG_AUTHENTICATOR_ASSERTION
    1. Create TAG_UAFV1_REG_ASSERTION
        1. Copy all the mandatory fields (see section TAG_UAFV1_REG_ASSERTION)
        2. If this is a first-factor roaming authenticator - add KeyID and key handle into internal storage
        3. If this is a bound authenticator - return key handle inside TAG_KEYHANDLE
    2. Put the entire TLV structure for TAG_UAFV1_REG_ASSERTION as the value of TAG_AUTHENTICATOR_ASSERTION
11. Return TAG_UAFV1_REGISTER_CMD_RESPONSE
    1. `UAF_CMD_STATUS_OK` as a status
    2. Add TAG_AUTHENTICATOR_ASSERTION
    3. Add TAG_KEY_HANDLE if the key handle must be stored outside the Authenticator

关于上面过程中，第1步有点疑问：ASM 向 Authenticator 传递 ASMRequest 时，其中的 KHAccessToken 已经包含了 AppID 了，为何这里还要 authenticator 再更新下？
在第5步由 authenticator 生成公私钥对 UAuth.pub/UAuth.priv
第6步，生成 RawKeyHandle，关于 RawKeyHandle 的结构可见【名词解释】
第7步，wrap这个RawKeyHandle。（这里wrap后就生成了加密后的KeyHandle，主要用于传给ASM）
第8步，生成 KRD(Create TAG_UAFV1_KRD structer)：

KRD 是包含在整个 authenticator 返回的 assertion 中的，assertion的结构如下（KRD结构应该在签名数据前都是）：

TLV Structure		Description
1	UINT16 Tag	TAG_UAFV1_REG_ASSERTION
1.1	UINT16 Length	Length of the structure
1.2	UINT16 Tag	TAG_UAFV1_KRD
1.2.1	UINT16 Length	Length of the structure
1.2.2	UINT16 Tag	TAG_AAID
1.2.2.1	UINT16 Length	Length of AAID
1.2.2.2	UINT8[] AAID	Authenticator Attestation ID
1.2.3	UINT16 Tag	TAG_ASSERTION_INFO
1.2.3.1	UINT16 Length	Length of Assertion Information
1.2.3.2	UINT16 AuthenticatorVersion	Vendor assigned authenticator version
1.2.3.3	UINT8 AuthenticationMode	For Registration this must be 0x01 indicating that the user has explicitly verified the action.
1.2.3.4	UINT16 SignatureAlgAndEncoding	Signature Algorithm and Encoding of the attestation signature. Refer to [UAFRegistry] for information on supported algorithms and their values.
1.2.3.5	UINT16 PublicKeyAlgAndEncoding	Public Key algorithm and encoding of the newly generated UAuth.pub key. Refer to [UAFRegistry] for information on supported algorithms and their values.
1.2.4	UINT16 Tag	TAG_FINAL_CHALLENGE
1.2.4.1	UINT16 Length	Final Challenge length
1.2.4.2	UINT8[] FinalChallenge	(binary value of) Final Challenge provided in the Command
1.2.5	UINT16 Tag	TAG_KEYID
1.2.5.1	UINT16 Length	Length of KeyID
1.2.5.2	UINT8[] KeyID	(binary value of) KeyID generated by Authenticator
1.2.6	UINT16 Tag	TAG_COUNTERS
1.2.6.1	UINT16 Length	Length of Counters
1.2.6.2	UINT32 SignCounter	Signature Counter. Indicates how many times this authenticator has performed signatures in the past.
1.2.6.3	UINT32 RegCounter	Registration Counter. Indicates how many times this authenticator has performed registrations in the past.
1.2.7	UINT16 Tag	TAG_PUB_KEY
1.2.7.1	UINT16 Length	Length of UAuth.pub
1.2.7.2	UINT8[] PublicKey	User authentication public key (UAuth.pub) newly generated by authenticator
1.3 (choice 1)	UINT16 Tag	TAG_ATTESTATION_BASIC_FULL
1.3.1	UINT16 Length	Length of structure
1.3.2	UINT16 Tag	TAG_SIGNATURE
1.3.2.1	UINT16 Length	Length of signature
1.3.2.2	UINT8[] Signature	Signature calculated with Basic Attestation Private Key over TAG_UAFV1_KRD content. The entire TAG_UAFV1_KRD content, including the tag and it's length field, MUST be included during signature computation.
1.3.3	UINT16 Tag	TAG_ATTESTATION_CERT (multiple occurrences possible) Multiple occurrences must be ordered. The attestation certificate MUST occur first. Each subsequent occurrence (if exists) MUST be the issuing certificate of the previous occurrence. The last occurence MUST be chained to one of the certificates included in field attestationRootCertificate in the related Metadata Statement [UAFAuthnrMetadata].
1.3.3.1	UINT16 Length	Length of Attestation Cert
1.3.3.2	UINT8[] Certificate	Single X.509 DER-encoded [ITU-X690-2008] Attestation Certificate or a single certificate from the attestation certificate chain (see description above).
1.3 (choice 2)	UINT16 Tag	TAG_ATTESTATION_BASIC_SURROGATE
1.3.1	UINT16 Length	Length of structure
1.3.2	UINT16 Tag	TAG_SIGNATURE
1.3.2.1	UINT16 Length	Length of signature
1.3.2.2	UINT8[] Signature	Signature calculated with newly generated UAuth.priv key over TAG_UAFV1_KRD content. The entire TAG_UAFV1_KRD content, including the tag and it's length field, MUST be included during signature computation.

因此这个KRD包含：AAID，ASSERTION_INFO（AuthenticatorVersion, AuthenticationMode, SignatureAlgAndEncoding(怎么计算出来的？)，PublicKeyAlgAndEncoding(怎么计算出来的？)），FINAL_CHALLENGE，KEYID，COUNTERS，PUB_KEY,（根据ebay源码，并不包括ATTESTATION_BASIC_FULL）

注意上面的TAG_ATTESTATION_CERT是会多次出现的，实际上是一个trust anchor而不是证书链，第一个一定是attestation certificate，后出现的证书要certificate前一个证书，最后出现的一定是 attestationRootCertificate中的证书。这个证书部分不知道怎么得来的

第9步，根据AttestationType去attestation KRD。这里说一下Full Basic Attestation：Authenticators must provide its attestation signature during the registration process.(这里没搞懂，是怎么一个过程：应该是最下面两个签名choice中的一个？第一个是用Attestation Private Key来签名KRD，第二个是用新生成的UAuth.priv key来签名KRD)

第10步，生成 TAG_AUTHENTICATOR_ASSERTION:

1. 生成上表的TAG_UAFV1_REG_ASSERTION
    1. 如果是一个first-factor roaming authenticator - add KeyID and key handle into internal storage.
    2. 如果是一个bound authenticator - return key handle inside TAG_KEYHANDLE(在TAG_UAFV1_REGISTER_CMD_RESPONSE - 这个是authenticator最终返回给ASM的报文).

第11步，生成authenticator最终返回给ASM的报文 TAG_UAFV1_REGISTER_CMD_RESPONSE：

	TLV Structure	Description
1	UINT16 Tag	TAG_UAFV1_REGISTER_CMD_RESPONSE
1.1	UINT16 Length	Command Length
1.2	UINT16 Tag	TAG_STATUS_CODE
1.2.1	UINT16 Length	Status Code Length
1.2.2	UINT16 Value	Status code returned by Authenticator
1.3	UINT16 Tag	TAG_AUTHENTICATOR_ASSERTION
1.3.1	UINT16 Length	Length of Assertion
1.3.2	UINT8[] Assertion	Registration Assertion (see section TAG_UAFV1_REG_ASSERTION).
1.4	UINT16 Tag	TAG_KEYHANDLE (optional)
1.4.1	UINT16 Length	Length of key handle
1.4.2	UINT8[] Value	(binary value of) key handle

上面的TAG_KEYHANDLE是可选的，根据第10步的策略

注意最后关于KeyID有这么一句：

NOTE

If the KeyID is generated randomly (instead of, for example, being derived from a key handle) - it should be stored inside RawKeyHandle so that it can be accessed by the authenticator while processing the Sign command.

这里是这么解释，RawKeyHandle的结构一般只包括：KHAccessToken, UAuth.priv, Username Size, Username，但是实际上，RawKeyHandle的结构是跟authenticator的具体实现有关，因此是可以放入KeyID的。

ASM拿到authenticator的response message后，将其传递给UAF Client，用于生成UAF Client的response报文：

The FIDO UAF Client MUST follow the steps:

1. Create a RegistrationResponse message
2. Copy RegistrationRequest.header into RegistrationResponse.header
3. Set RegistrationResponse.fcParams to FinalChallenge (base64url encoded serialized and utf8 encoded FinalChallengeParams)
4. Append the response from each Authenticator into RegistrationResponse.assertions
5. Send RegistrationResponse message to FIDO Server

一个例子如下：

EXAMPLE 8: Registration Response


```
[{
    "assertions": [
      {
        "assertion": "AT7uAgM-sQALLgkAQUJDRCNBQkNEDi4HAAABAQEAAAEKLiAA9t
BzZC64ecgWQBGSQb5QtEIPC8-Vav4HsHLZDflLaugJLiAAZMCPn92yHv1Ip-iCiBb6i4ADq6
ZOv569KFQCvYSJfNgNLggAAQAAAAEAAAAMLkEABJsvEtUsVKh7tmYHhJ2FBm3kHU-OCdWiUY
VijgYa81MfkjQ1z6UiHbKP9_nRzIN9anprHqDGcR6q7O20q_yctZAHPjUCBi5AACv8L7Y1RM
x10gPnszGO6rLFqZFmmRkhtV0TIWuWqYxd1j00wxam7i5qdEa19u4sfpHFZ9RGI_WHxINkH8
FfvAwFLuOBMIIB6TCCAY8CAQEwCQYHKoZIzj0EATB7MQswCQYDVQQGEwJVUzELMAkGA1UECA
wCQ0ExCzAJBgNVBAcMAlBBMRAwDgYDVQQKDAdOTkwsSW5jMQ0wCwYDVQQLDARBQU4xMRMwEQ
YDVQQDDApOTkwsSW5jIENBMRwwGgYJKoZIhvcNAQkBFg1ubmxAZ21haWwuY29tMB4XDTE0MD
gyODIxMzU0MFoXDTE3MDUyNDIxMzU0MFowgYYxCzAJBgNVBAYTA1VTMQswCQYDVQQIDAJDQT
EWMBQGA1UEBwwNU2FuIEZyYW5jaXNjbzEQMA4GA1UECgwHTk5MLEluYzENMAsGA1UECwwERE
FOMTETMBEGA1UEAwwKTk5MLEluYyBDQTEcMBoGCSqGSIb3DQEJARYNbm5sQGdtYW1sLmNvbT
BZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCGBt3CIjrDowzSiF68C2aErYXnDUsWXOYxqIP
imOOWg9FFdUYCa6AgKjn1R99Ek2d803sGKROivnavmdVH-SnEwCQYHKoZIzj0EAQNJADBGAi
EAzAQujXnSS9AIAh61Gz6ydypLVTsTnBzqGJ4ypIqy_qUCIQCFsuOBGcRV-o4GHPBph_VMrG
3NpYh2GKPjsAim_cSNmQ",
        "assertionScheme": "UAFV1TLV"
      }
    ],
    "fcParams": "eyJhcHBJRCI6Imh0dHBzOi8vdWFmLXRlc3QtMS5ub2tub2t0ZXNOLmN
vbTo4NDQzL1NhbXBsZUFwcC91YWYvZmFjZXRzIiwiY2hhbGxlbmdlIjoiSDlpVzl5QTlhQVh
GX2xlbFFvaV9EaFVrNTE0QWQ4VHF2MHpDbkNxSORwbyIsImNoYW5uZWxCaW5kaW5nIjp7fSw
iZmFjZXRJRCI6ImNvbS5ub2tub2suYW5kcm9pZC5zYW1wbGVhcHAifQ",
    "header": {
 "appID": "https://uaf-test-1.noknoktest.com:8443/SampleApp/uaf/facets",
      "op": "Reg",
      "serverData": "IjycjPZYiWMaQ1tKLrJROiXQHmYG0tSSYGjP5mgjsDaM17RQgq0
dl3NNDDTx9d-aSR_6hGgclrU2F2Yj-12S67v5VmQHj4eWVseLulHdpk2v_hHtKSvv_DFqL4n
2IiUY6XZWVbOnvg",
      "upv": {
        "major": 1,
        "minor": 0
      }
    }
}]
```

FIDO Server解析UAF Client发回来的response：

The FIDO Server MUST follow the steps:

1. Parse the message
    1. If protocol version (`RegistrationResponse.header.upv`) is not supported – reject the operation
    2. If a mandatory field in UAF message is not present or a field doesn't correspond to its type and value - reject the operation

2. Verify that `RegistrationResponse.header.serverData`, if used, passes any implementation-specific checks against its validity. See also section ServerData and KeyHandle.
3. base64url decode `RegistrationResponse.fcParams` and convert it into an object (`fcp`)
4. Verify each field in fcp and make sure it is valid:
    1. Make sure `fcp.appID` corresponds to the one stored by the FIDO Server
    2. Make sure `fcp.challenge` has really been generated by the FIDO Server for this operation and it is not expired
    3. Make sure `fcp.facetID` is in the list of trusted FacetIDs [FIDOAppIDAndFacets]
    4. Make sure `fcp.channelBinding` is as expected (see section ChannelBinding dictionary)
    5. Reject the response if any of these checks fails
5. For each assertion `a` in `RegistrationResponse.assertions`
    1. Parse TLV data from `a.assertion` assuming it is encoded according to the suspected assertion scheme `a.assertionScheme` and make sure it contains all mandatory fields (indicated in Authenticator Metadata) it is supposed to have and has a valid syntax.
        - If it doesn't - continue with next assertion
    2. Retrieve the AAID from the assertion.

        > NOTE
        >
        > The AAID in `TAG_UAFV1_KRD` is contained in `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_AAID`.

    3. Verify that `a.assertionScheme` matches `Metadata(AAID).assertionScheme`
        - If it doesn't match - continue with next assertion
    4. Verify that the AAID indeed matches the policy specified in the registration request.

        > NOTE
        >
        > Depending on the policy (e.g. in the case of AND combinations), it might be required to evaluate other assertions included in this `RegistrationResponse` in order to determine whether this AAID matches the policy.

        - If it doesn't match the policy - continue with next assertion
    5. Locate authenticator-specific authentication algorithms from the authenticator metadata [UAFAuthnrMetadata] using the AAID.
    6. Hash `RegistrationResponse.fcParams` using hashing algorithm suitable for this authenticator type. Look up the hash algorithm in authenticator metadata, field `AuthenticationAlgs`. It is the hash algorithm associated with the first entry related to a constant with prefix UAF_ALG_SIGN.
        - `FCHash = hash(RegistrationResponse.fcParams)`
    7. if `a.assertion` contains an object of type `TAG_UAFV1_REG_ASSERTION`, then
        1. if `a.assertion.TAG_UAFV1_REG_ASSERTION` contains `TAG_UAFV1_KRD` as first element:
            1. Obtain `Metadata(AAID).AttestationType` for the AAID and make sure that `a.assertion.TAG_UAFV1_REG_ASSERTION` contains the most preferred attestation tag specified in field `MatchCriteria.attestationTypes` in `RegistrationRequest.policy` (if this field is present).
                - If `a.assertion.TAG_UAFV1_REG_ASSERTION` doesn't contain the preferred attestation - it is RECOMMENDED to skip this assertion and continue with next one
            2. Make sure that `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.FinalChallenge` == `FCHash`
                - If comparison fails - continue with next assertion
            3. Obtain `Metadata(AAID).AuthenticatorVersion` for the AAID and make sure that it is lower or equal to `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.AuthenticatorVersion`.
                - If `Metadata(AAID).AuthenticatorVersion` is higher (i.e. the authenticator firmware is outdated), it is RECOMMENDED to assume increased risk. See sections "StatusReport dictionary" and "Metadata TOC object Processing Rules" in [UAFMetadataService] for more details on this.
            4. Check whether `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.RegCounter` is acceptable, i.e. it is either not supported (value is 0) or it is not exceedingly high
                - If `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.RegCounter` is exceedingly high, this assertion might be skipped and processing will continue with next one
            5. If `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD` contains `TAG_ATTESTATION_BASIC_FULL` tag
                1. If entry `AttestationRootCertificates` for the AAID in the metadata [UAFAuthnrMetadata] contains at least one element:
                    1. Obtain contents of all `TAG_ATTESTATION_CERT` tags from `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_ATTESTATION_BASIC_FULL` object. The occurrences are ordered (see [UAFAuthnrCommands]) and represent the attestation certificate followed by the related certificate chain.
                    2. Obtain all entries of `AttestationRootCertificates` for the AAID in authenticator Metadata, field `AttestationRootCertificates`.
                    3. Verify the attestation certificate and the entire certificate chain up to the Attestation Root Certificate using Certificate Path Validation as specified in [RFC5280]
                        - If verification fails – continue with next assertion
                    4. Verify `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_ATTESTATION_BASIC_` the attestation certificate (obtained before).
                        - If verification fails – continue with next assertion
                2. If `Metadata(AAID).AttestationRootCertificates` for this AAID is empty - continue with next assertion
                3. Mark assertion as positively verified

6. If `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD` contains an object of type `TAG_ATTESTATION_BASIC_SURROGATE`
    1. There is no real attestation for the AAID, so we just assume the AAID is the real one.
    2. If entry `AttestationRootCertificates` for the AAID in the metadata is empty
        - Verify `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_ATTESTATION_BASIC_SURROGATE.Signa`
            - If verification fails – continue with next assertion
    3. If entry `AttestationRootCertificates` for the AAID in the metadata is not empty - continue with next assertion (as the AAID obviously is expecting a different attestation method).
    4. Mark assertion as positively verified
7. If `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD` contains another `TAG_ATTESTATION` tag - verify the attestation by following appropriate processing rules applicable to that attestation. Currently this document only defines the processing rules for Basic Attestation.

2. if `a.assertion.TAG_UAFV1_REG_ASSERTION` contains a different object than `TAG_UAFV1_KRD` as first element, then follow the rules specific to that object.
3. Extract `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.PublicKey` into PublicKey, `a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.KeyID` into KeyID,`a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.SignCounter` into SignCounter,`a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_ASSERTION_INFO.authenticatorV` AuthenticatorVersion,`a.assertion.TAG_UAFV1_REG_ASSERTION.TAG_UAFV1_KRD.TAG_AAID` into AAID.

8. if `a.assertion` doesn't contain an object of type `TAG_UAFV1_REG_ASSERTION`, then skip this assertion (as in this UAF v1 only `TAG_UAFV1_REG_ASSERTION`is defined).

6. For each positively verified assertion `a`
    - Store PublicKey, KeyID, SignCounter, AuthenticatorVersion, AAID and `a.tcDisplayPNGCharacteristics` into a record associated with the user's identity . If an entry with the same pair of AAID and KeyID already exists then fail (should never occur).