DECTA Secure for Acquirers

Overview

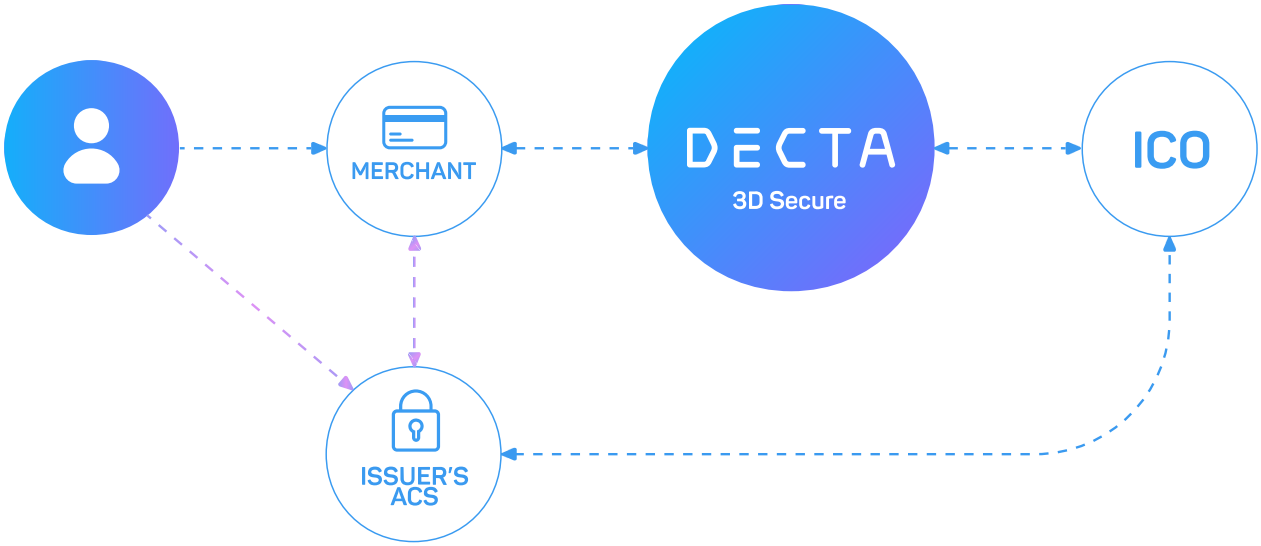
Preventing payment fraud and reducing checkout friction are essential for business to provide a secure and convenient digital shopping experience for the customers. To achieve this, payment card issuers and merchants use EMV 3DS to seamlessly authenticate consumers and protect against card-not-present (CNP) fraud. DECTA 3D Secure for Acquirers is a product designed for PSPs, acquirers, and merchants who want to comply with EMV 3DS in the most advanced way possible, without sacrificing security and performance.

The DECTA 3D Secure for Acquirers service supports all existing and available versions of EMV 3DS for Mastercard and Visa card schemes. The service also supports various Strong Consumer Authentication (SCA) exemptions, as well as 3RI and APP device channels.

To support the implementation of the DECTA 3D Secure service, the detailed technical documentation is provided prior to integration.

DECTA 3D Secure for Acquirers service

The DECTA 3D Secure for Acquirers service is based on XML POST requests. When a merchant submits an initial request, the MPI service sends EMV 3D-specific requests to the ICO to obtain the card status, issuer ACS server address, and desired authentication flow. The MPI service never receives the user's browser session. Instead, the payment page application has access to the user session, performs any necessary redirections, and receives the response. This ensures that the cardholder's data is not exposed to anyone other than the merchant and the card issuing bank.



DECTA 3D Secure schema

Regular Authentication Challenge Flow

Only steps directly related to the end-user and the merchant are numbered. Other EMV 3D specific steps are performed by the DECTA MPI service.

Step 1 - The merchant's payment page requests the cardholder to provide the necessary payment data, such as card number, expiry, etc.

Step 2 - The merchant's payment page creates an Enrollment Request (initial) and calculates the signature of the Message element to be posted to the DECTA MPI service. The signature is then set to the DECTA MPI service message which is posted to the DECTA MPI XML API with Content-Type=application/xml.

> It is important to note that the merchant should set the `termUrl` parameter in the request message to provide a Merchant page URL, where ACS will post (via user browser) the CRes message after cardholders authentication.

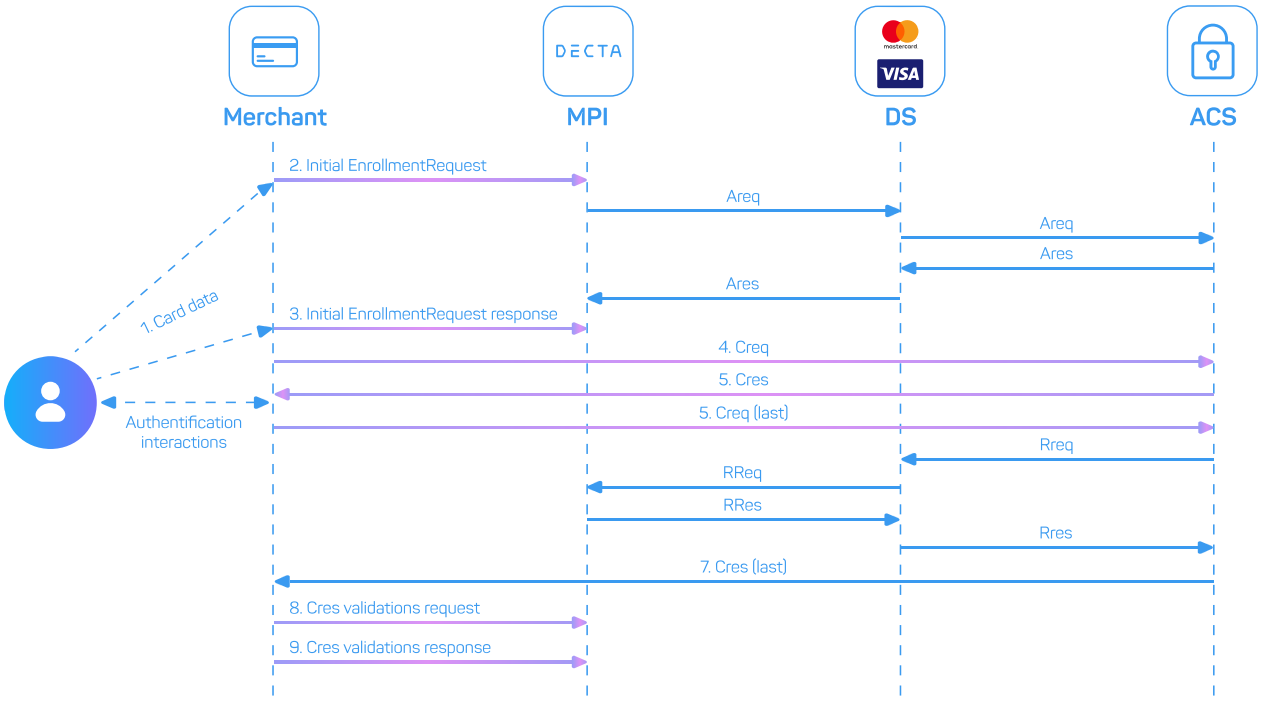
Step 3 - The DECTA MPI service checks the card participation from the 3D Secure directory published ranges:

- If there are no errors or no challenges requested, the response message will contain the `RedirectToACSForm` field with the information that needs to be sent to the cardholder's browser to initiate a CReq request to the ACS for authentication;
- With error or final (frictionless) authentication results, the 3DS flow ends. Based on the results, the merchant determines whether to proceed authorisation or not.

Step 4-6 - `RedirectToACSForm` field value is opened in cardholder's browser and it performs authentication steps defined by the Issuer's ACS server.

Step 7 - After ACS finishes with the cardholder's authentication, it returns control to the merchant by receiving the CRes to the termUrl (set in initial request).

Step 8-9 - Merchant sends a `PAResValidationRequest` to the MPI service with the received CRes content. The MPI service verifies the CRes and matches it with the appropriate RReq from the directory (ICO), and responds to the merchant with the final outcome. Based on the returned parameters, the merchant proceeds with transaction authorization or stops the transaction in case of an error or authentication failure.



Regular authentication challenge flow

DECTA MPI XML Signature Example

Example code for Python

```
         from signxml import XMLSigner
        from lxml import etree

        with open('cert.pem', 'r') as f:
            cert_pub = f.read()
        with open('key.pem', 'r') as f:
            key_priv = f.read()

        document = etree.parse('document.xml')
        for modirummpi in document.findall('{http://www.modirum.com/schemas/mpiapi}
Message'):
            message_id = modirummpi.get('messageId')

        reference_uri = f'#{message_id}'
        id_attribute = 'messageId'
        signed_envelope = XMLSigner(
            c14n_algorithm='http://www.w3.org/TR/2001/REC-xml-
c14n-20010315').sign(document, key=key_priv,cert=cert_pub,reference_uri=reference_uri,id_at
tribute=id_attribute)
        file = open("signed.xml", "w")
        file.write(etree.tostring(signed_envelope, encoding=str))
        file.close()
```