EMV®
3-D Secure

SDK Technical Guide

Version 2.1.0
October 2017

Legal Notice

Contents

Figures

Tables

1 Introduction

1.1 About EMV 3-D Secure SDK Technical Guide

This document is the technical guide of the EMV 3-D Secure SDK. Its purpose is to give additional insight to the implementation of the described SDK based on the specification documentation.

EMV 3-D Secure Core Protocol and Specification describes how authentication transactions would happen in a mobile application environment. From an implementation perspective, the merchant's mobile application is separated from the 3-D Secure SDK (Software Development Kit that is also referred to as "SDK" in this document), which would implement most of the 3-D Secure functionality (collection of device data, communication towards the ACS, and user authentication) in the mobile device. Furthermore, the Core Specification describes the functionality and requirements for the 3DS Requestor in general.

During the spring and summer of 2016, EMVCo conducted a project where an SDK was implemented as proof-of-concept based on the newly-created 3-D Secure specifications. The implementation and screen captures appearing in this document are from the aforementioned project.

While this document discusses the SDK in the context of a mobile device, similar SDK can be implemented for other devices such as smart TV's and tablets. Furthermore, this document may refer to the 3DS Requestor as "Merchant" and talk about "Merchant Application". In addition, the document also uses the term "Purchase transaction" to describe the whole transaction that the cardholder is performing. This is just for clarity reasons and not intended to limit the transactions to those dealing with a purchase. For a more precise terminology, please see the Core Specification page.

If not otherwise stated, this document uses screen captures from an Android implementation of the SDK.

1.2 Overview and Scope

This document describes the implementation aspect of 3-D Secure SDK. The purpose of this document is to provide insight on implementation of an SDK, not to serve as an explicit guide. The examples and code samples presented in this document are meant to give guidance and overview on how a certain functionality e.g. encryption can be implemented.

The following diagram describes the bigger picture of the environment that the SDK is part of.

On a high level, the SDK's functionality consists of the following:

- Initialization of the transaction and receiving required information from the 3DS Requestor App
- Collection of device data, encrypting it, and returning the encrypted data to 3DS Requestor App
- Handling of the user interface (challenge-related views, providing the progress view object to 3DS Requestor App) during the transaction based on the given parameters and information (UICustomization, CRes)
- Perform the challenge step of the 3-D Secure transaction

- Secure and encrypted communication between SDK and ACS (CReq/CRes, challenge)

In a 3-D Secure transaction, there are parts that the SDK does not cover. As the 3DS Requestor App starts the authentication transaction, it communicates directly with the 3DS Requestor and 3DS Server. This communication is not in the scope of the SDK and thus not in this document.

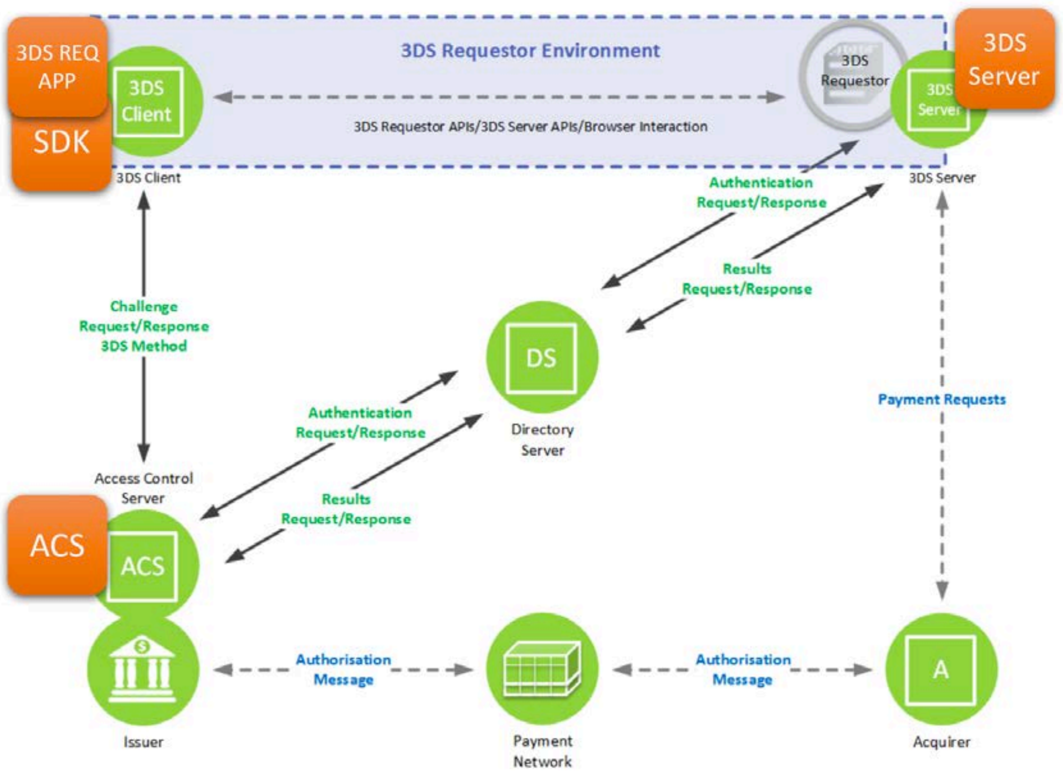
1.3 Other systems described in this document

The SDK is a component in the 3DS Requestor Environment and it interfaces with other 3-D Secure systems. The figure below illustrates the big picture of 3-D Secure and how the SDK fits in the big picture.

For more detailed information on the topic, see the chapter 2 of the Core Specification.

The systems that are relevant from the SDK's viewpoint are marked with orange color and described below.

Figure 1 3-D Secure components



3DS Requestor App

3DS Requestor App is a mobile application in the cardholder's mobile device. The 3DS Requestor App has the SDK in it to perform the 3-D Secure functionality.

For more information about the 3DS Requestor App, see chapter 5 Merchant Implementation .

3DS Server

The 3DS Server is a part of the 3DS Requestor environment that implements the 3-D Secure protocol and functionality. Its main purpose is to act as a Merchant side component of the transaction, communicating towards Directory Server and ACS (AReq/ARes and RReq/RRes) and providing the 3DS Requestor App with the required information to perform the transaction (transaction status, does the transaction require challenge, needed information for SDK to proceed with CReq/CRes).

The communication between the 3DS Requestor App and the 3DS Server is not in the scope of 3-D Secure or this document. Each vendor may implement their own proprietary communication between the two components as long as it meets the security requirements described in the Core Specification.

ACS

The ACS is the only 3-D Secure component that the SDK communicates with. ACS serves as the Issuer side component implementing the 3-D Secure protocol.

From an SDK viewpoint, the ACS performs the challenge functionality and communicates with the SDK using CReq and CRes messages. The communication between the SDK and ACS is secured and encrypted.

For more details on ACS in 3-D Secure transactions, see the Core Specification.

1.4 Audience

This document is intended for use by Implementers who want to develop a 3-D Secure SDK. This document gives examples of the implementation and should be read together with 3-D Secure Core Specification and 3-D Secure SDK Specification

1.5 Related documents

The reader should be familiar with the following documents that this document refers to and the 3-D Secure protocol and transaction flow in general.

The documentation below is referred as “Specifications”.

EMV 3-D Secure Protocol and Core Specification

The EMV 3-D Secure Protocol and Core Specification (referred to as “Core Specification”) describes the details of the 3-D Secure protocol, infrastructure, components, and specifies the requirements for each component and their interaction.

EMV 3-D Secure SDK Specification

The EMV 3-D Secure SDK Specification (version 2.1.0, referred to as “SDK Specification”) describes the requirements and functionality of the 3-D Secure SDK. The document has detailed descriptions of the classes and methods that the SDK offers to the 3DS Requestor App to perform its tasks.

EMV 3-D Secure SDK – Device information

The EMV 3-D Secure SDK – Device Information describes the device identification parameters that shall be collected by the 3DS SDK.

1.6 Notational Conventions

1.6.1 Definitions

For the definition of the terms used in this specification, refer to Table 1.3: Definitions in the EMV 3-D Secure Protocol and Core Functions Specification.

1.6.2 Abbreviations

This document uses the same abbreviations as the Core Specification.

Table 1 Abbreviations

Abbreviation	Description
3DS	Three Domain Secure
3DS SDK	Three Domain Secure Software Development Kit
ACS	Access Control Server
AReq	Authentication Request
ARes	Authentication Response
AV	Authentication Value
BIN	Bank Identification Number
CA	Certificate Authority
CA DS	Certificate Authority Directory Server
CReq	Challenge Request
CRes	Challenge Response
DS	Directory Server
ECI	Electronic Commerce Indicator

Abbreviation	Description
JSON	JavaScript Object Notation
MAC	Message Authentication Code
NPA	Non-Payment Authentication
NVP	NameValuePair
OOB	Out-of-Band
PA	Payment Authentication
OTP	One-time Passcode
SDK	Software Development Kit
TLS	Transport Layer Security
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

1.6.3 Terminology and Conventions

The following words are used in this guide and have a specific meaning:

Purchase transaction

The term is used to describe the full transaction cycle from user’s viewpoint and is not intended to limit the transactions to ones where a purchase is involved.

Merchant

The term is used to refer to the 3DS Requestor. For clarity reasons and as an example of a 3DS Requestor, the term Merchant may be used.

2 Implementation of the transaction flows

This chapter describes the implementation of the SDK from the 3-D Secure transaction viewpoint. The content has been divided into chapters based on the flow, see the left-hand side for the phases.

The implementation decisions documented in this chapter are indicative in nature and originate from the proof-of-concept implementation project conducted by EMVCo 3-D Secure Task Force in spring and summer of 2016. There may be other ways to implement the specified functionality.

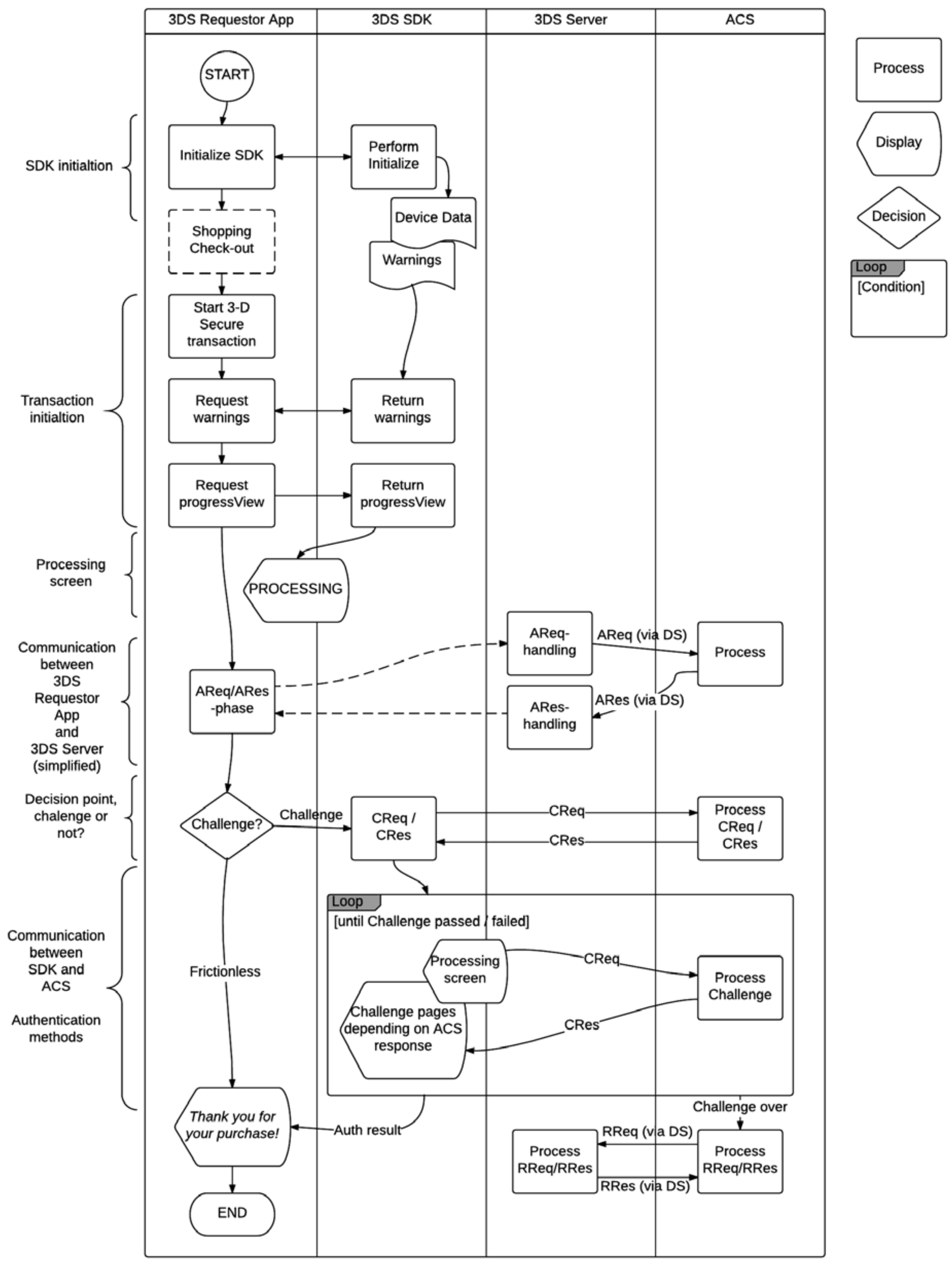
The 3-D Secure transaction is defined in more detail in the Core Specification and SDK Specification documents. This document references to the Core Specification where necessary.

This chapter is organized to follow the transaction flow. The flow has been divided into the following chapters:

- Chapter 2.1 SDK initiation describes the steps that the SDK takes when the 3DS Requestor App is started and the SDK initiated
- Chapter 2.2 Frictionless flow describes the steps that the 3DS Requestor App and the SDK take during a frictionless flow. Note that parts of the 3-D Secure transaction take place outside the SDK.
- Chapter 2.3 Challenge flow describes the steps that the 3DS Requestor App and the SDK take to perform the rest of the transaction if a challenge is required.

Figure 2 3-D Secure transaction below illustrates the transaction flow on a high level.

Figure 2 3-D Secure transaction



2.1 SDK initiation

The Initiation phase is executed when the 3DS Requestor App is started. It calls the `initialize()` –method of the SDK which performs relevant tasks and changes its status to `Initialized`.

The `initialize()`-method performs the following tasks:

- SDK settings are loaded (configuration parameters, locale, UI customization)
- Security checks are performed (see list of security requirements in Core Specification)
- Device data is collected for all protocol versions that the SDK supports (see 3-D Secure SDK – Device Information document)
- Possible warnings are generated. The 3DS Requestor App may request them when needed.

After a successful initialization, the SDK is ready to perform 3-D Secure transactions.

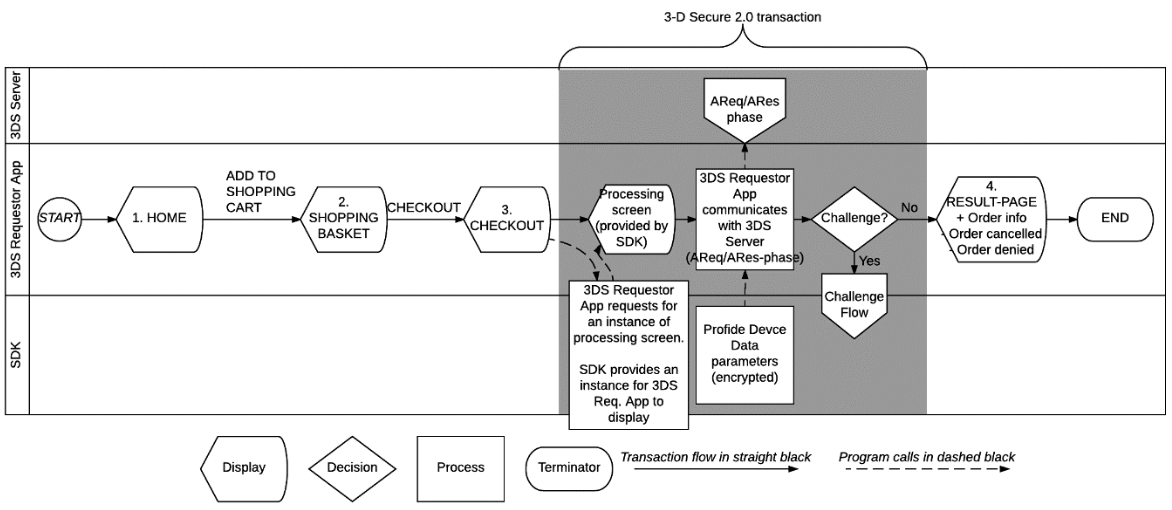
For more information on the security checks performed during the initialization phase, see chapter 3.5 Implementation of Security .

2.2 Frictionless flow

This chapter describes the transaction flow from start to the point where the 3DS Requestor App has the information to decide if a challenge is needed (status "C" in ARes-message) or not.

If challenge is not needed, the 3DS Requestor App calls the SDK only in the beginning when the transaction is initiated and an instance of `progressView` is provided to the 3DS Requestor App by the SDK.

Figure 3 Simplified transaction flow (Frictionless)


3-D Secure 2.0 transaction
3DS Server
3DS Requestor App
SDK
AReq/ARes phase
START
1. HOME
ADD TO SHOPPING CART
2. SHOPPING BASKET
CHECKOUT
3. CHECKOUT
Processing screen (provided by SDK)
3DS Requestor App communicates with 3DS Server (AReq/ARes-phase)
Challenge?
No
Yes
4. RESULT-PAGE + Order info Order cancelled Order denied
END
Challenge Flow
3DS Requestor App requests for an instance of processing screen.
SDK provides an instance for 3DS Req. App to display
Profide Devce Data parameters (encrypted)
Display
Decision
Process
Terminator
Transaction flow in straight black
Program calls in dashed black

2.2.1 Transaction initiation

The transaction can be initiated by the 3DS Requestor App once it has all the necessary information needed to perform the transaction. As the 3DS Requestor App initiates the 3-D Secure transaction, it should perform the following tasks:

- Transaction instance is created (`createTransaction()`-method)
- Device Data is collected and returned to the 3DS Requestor App
- Return an instance of `progressView` to the 3DS Requestor App (`getProgressView()`-method)

  Do note that the Core Specification does not specify that it is the responsibility of the SDK to gather the needed information. The information that the 3DS Server needs to construct and process the AReq-message must come from a component within the 3DS Requestor Environment.

2.2.2 Device Data collection

The SDK collects device data during the transaction initiation phase. All the device data required for all protocol versions supported by the SDK has to be collected during the ThreeDS2Service initialization. However, only the device data fields for the selected protocol version of the transaction should be included in the transaction `getDeviceData()` method. This `getDeviceData()` method returns the Device Data as an encrypted String. The Device Data is intended to be encrypted through the 3DS Requestor Environment and decrypted when it reaches the Directory Server in the AReq-message.

The device data parameters are described in more detail in the EMV 3-D Secure SDK Device Information document.

Note that terms Device Information and Device Parameters are also used for this purpose.

Permissions and Device Data collection

Note that the SDK may use only those permissions that the 3DS Requestor Application has requested. The SDK itself shall not request additional permissions.

The SDK Specification states that there should not be any run-time prompting of permissions when the SDK performs its operations. All permissions should be granted to the application during installation or before the SDK is initiated.

Collection of device data must be implemented so that the process is not stopped even though the SDK does not have the permission to collect the information, e.g. location.

2.2.3 Progress View

The processing of the 3-D Secure transaction takes a few seconds. For the user experience to remain smooth, a Progress View has been specified to be displayed while the 3DS Requestor App or the SDK perform tasks in the background.

The progress view is generated by the SDK. The 3DS Requestor App requests the SDK for an instance of it and displays it before starting communication to the 3DS Server (authentication request, AReq/ARes-phase).

See chapter 4.2.1 for a sample implementation of the Progress View.

2.2.4 Authentication request (i.e. the AReq/ARes-phase)

This step happens during a 3-D Secure transaction but is not specified in the Specifications. The AReq/ARes-phase is a part of the 3-D Secure transaction but not in scope of the SDK. The 3DS Requestor App communicates with the 3DS Requestor Environment, which handles the phase.

The SDK's only role in this phase has to do with the device data collected in initiation phase. 3DS Requestor App receives the device data parameters from the SDK in an encrypted form. The device data is sent by the 3DS Requestor App to 3DS Server for AReq/ARes-processing.

As this phase is out of the SDK's scope and implemented by the 3DS Requestor App, see the Merchant Implementation in chapter 5 for more details.

2.2.5 Decision point → Frictionless flow vs. Challenge flow

Based on the information received by the 3DS Requestor Server after the AReq/ARes-phase, the 3DS Requestor App may end the authentication transaction (Frictionless flow, status Y, A or error) or continue the transaction to Challenge (status "C" in ARes-message).

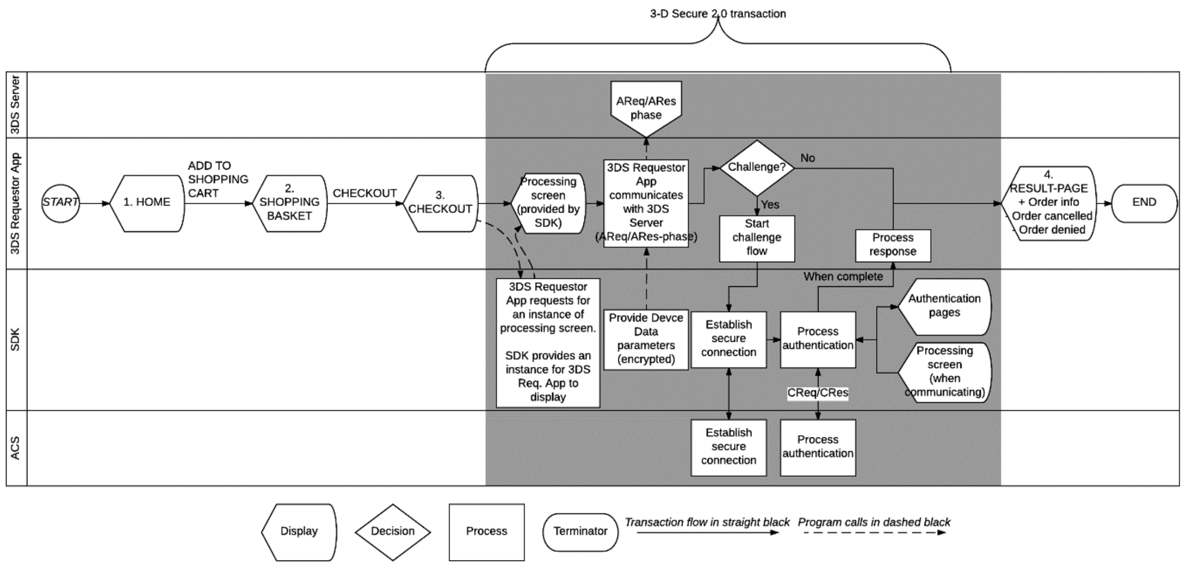
If the 3-D Secure transaction is regarded as "Frictionless" (status anything but "C" in the ARes-message), the processing ends here and the SDK is not called. The 3DS Requestor App proceeds with the purchase transaction based on the information it has received from the 3DS Requestor Server.

If the 3-D Secure transaction requires a challenge (status "C" in ARes-message), the 3DS Requestor App calls the SDK to perform the challenge phase.

2.3 Challenge flow

From transaction viewpoint, the Challenge flow is continuation of the Frictionless flow in transactions where a challenge is needed (status "C" in ARes-message).

Figure 4 Simplified transaction flow (Challenge)


3-D Secure 2.0 transaction
3DS Server
3DS Requestor App
SDK
ACS
AReq/ARes phase
START
1. HOME
ADD TO SHOPPING CART
2. SHOPPING BASKET
CHECKOUT
3. CHECKOUT
Processing screen (provided by SDK)
3DS Requestor App communicates with 3DS Server (AReq/ARes-phase)
Challenge?
No
Yes
Start challenge flow
Process response
4. RESULT-PAGE + Order info Order cancelled Order denied
END
When complete
3DS Requestor App requests for an instance of processing screen. SDK provides an instance for 3DS Req. App to display
Provide Devce Data parameters (encrypted)
Establish secure connection
Process authentication
Authentication pages
Processing screen (when communicating)
CReq/CRes
Establish secure connection
Process authentication
Display
Decision
Process
Terminator
Transaction flow in straight black
Program calls in dashed black

2.3.1 Communication between SDK and ACS

SDK establishes a secure connection to the ACS. The secure connection details are defined in detail in the Core Specification (Chapters 6.1.4 and 6.2.3)

- The communication between the SDK and ACS is encrypted. For more information on the encryption and code samples, see chapter 3 Security and cryptography in this document.

2.3.2 Challenge dialogue

The challenge dialogue is the most visible part of the SDK. The Core Specification defines a set of pages with their own purpose. The SDK selects the appropriate display pages based on what the CRes-message to be used in the challenge dialogue.

The SDK implements two UI-types:

- Native pages are implemented with the native UI components of the platform
- HTML pages are implemented as a WebView component, displaying the HTML content sent by ACS

Page types

The SDK selects the page type depending on what the ACS has requested in CRes-message (ACS UI Type).

- Text
- Single select
- Multi select
- Out of Band
- HTML

SDK renders the page type with information provided by the ACS.

Note about Out of Band

In Out of Band channel the authentication itself happens in a separate application or device.

2.4 Namespace

Vendor should have the EMV 3DS functionality in their own namespace according to platform conventions.

2.4.1 iOS

In iOS Objective-C code this means prefixing the names according to Apple conventions described in Apple developer documentation.

For more information on the topic, see https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Conventions/Conventions.html

2.4.2 Other platforms

On other platforms this means using packages that belong to vendor's namespace. For more details, see platform specific documentation and conventions on namespaces.

3 Security and cryptography

This chapter gives examples and code samples regarding the security and cryptography implementations. There are many ways to implement the functionality, the attached code samples show one way of doing it. The samples are from the proof-of-concept implementation of the EMV 3-D Secure SDK.

3.1 Device data encryption

Device data encryption is defined in the Core Specification in chapter 6.2.2.1. Core Specification defines the supported algorithms as RSA-OAEP and encryption as A128CBC-HS256 or A128GCM. SDK does the encryption with the directory server specific public key shipped in the app itself. Security of the stored data is discussed in chapter 3.5.14

Encryption of stored data.

Here are examples of encryption on different platforms.

3.1.1 Android

Proof-of-concept implementation uses Nimbus JOSE JWE library (http://connect2id.com/products/nimbus-jose-jwt). This library implements everything needed for device data encryption on SDK with reasonably high level interface. Example code can be found below.

If the DS public key is an RSA key:

```
public static String jweEncrypt(String dd, RSAPublicKey publicKey) {
        try {
            JWTClaimsSet set = JWTClaimsSet.parse(dd);
            EncryptedJWT jwt = new EncryptedJWT(
                    new JWEHeader(JWEAlgorithm.RSA_OAEP,
            EncryptionMethod.A128CBC_HS256),
                    set);

            jwt.encrypt(new RSAEncrypter(publicKey));

            String result = jwt.serialize();
            return result;
        } catch (Exception e) {
    }
        return null;
}
```

Else, if the DS public key is an EC key:

```
public static String jweEncrypt(String dd, ECPublicKey publicKey, String
dsId) {
         KeyPair kp = null;
         try {
            JWTClaimsSet set = JWTClaimsSet.parse(dd);
        kp = Crypto.generateEphemeralKeyPair();
         SecretKey k = Crypto.generateECDHSecret(publicKey,
            (ECPrivateKey)kp.getPrivate(), dsId);
    JWK jwk = new ECKey.Builder(ECKey.Curve.P_256,
            (ECPublicKey)kp.getPublic()).build();

       JWEHeader header = new JWEHeader.Builder(JWEAlgorithm.DIR,
            EncryptionMethod.A128CBC_HS256).
                    ephemeralPublicKey(ECKey.parse(jwk.toJSONString()))
                    .build();

            JWEObject jweObject = new JWEObject(header, new Payload(dd));
            jweObject.encrypt(new DirectEncrypter(k));

            String result = jweObject.serialize();
            return result;
        } catch (Exception e) {
        } finally {
            kp = null;
        }
        return null;
}
```

3.1.2 iOS 9

On iOS 9 platform proof-of-concept implementation does the JWE calculation directly using cryptography primitives available from commonCrypto libraries. The algorithm closely follows the description in RFC 7516. All needed components are generated, and then the actual JWE structure is filled according to the RFC.

If the DS public key is an RSA key:

```
/* Device Data encryption - create JWE given DS publicKey */
+(NSString *)createJWE:(NSString *)payload
withPublicKey:(SecKeyRef)publicKey {
  // create secretKey for encryption
  NSData *secret = [self generateRandom:(KEY_SIZE*2)];
  NSData *hmacKey  = [secret subdataWithRange:NSMakeRange(0, KEY_SIZE)];
  NSData *aesKey = [secret subdataWithRange:NSMakeRange(KEY_SIZE,
KEY_SIZE)];
  NSData *iv = [self generateRandom: IV_SIZE];

  // create header
  NSString *header = @"{\"enc\":\"A128CBC-HS256\",\"alg\":\"RSA-OAEP\"}";

  // encrypt secretKey
  NSData *encryptedKey = [self rsaEncrypt:secret key:publicKey];

  // encrypt payload
  NSData *encrypted = [self aesEncrypt:[payload
dataUsingEncoding:NSUTF8StringEncoding] withKey:aesKey withIV:iv];
  NSString *basePayload = [encrypted unpaddedBase64URLEncoded];
  NSString *baseCEK = [encryptedKey unpaddedBase64URLEncoded];
  NSString *baseHeader = [[header dataUsingEncoding:NSUTF8StringEncoding]
unpaddedBase64URLEncoded];
  NSString *baseIV = [iv unpaddedBase64URLEncoded];
  // create auth hash
  NSData *hmac = [self hmac: encrypted withKey: hmacKey withIV: iv withA:
[baseHeader dataUsingEncoding:NSASCIIStringEncoding]];

  return [NSString stringWithFormat:@"%@.%@.%@.%@.%@", baseHeader, baseCEK,
baseIV, basePayload, [[self hmacToTag: hmac] unpaddedBase64URLEncoded]];
}
```

Else, if the DS public key is an EC key:

```
/* Device Data encryption - create JWE given DS publicKey */
+(NSString *) createJWE:(NSData *)payload
             withSecret: (NSData *)secret
          withPublicKey:(NSString *)jwk {
  // create header
  NSString *header = [NSString stringWithFormat:
@"{\"enc\":\"A128CBC-HS256\",\"alg\":\"dir\",\"epk\":%@ }",jwk];
  NSLog(@"Header %@", header);

  // encrypt payload
  NSData *hmacKey  = [secret subdataWithRange:NSMakeRange(0,
KEY_SIZE)];
  NSData *aesKey = [secret subdataWithRange:NSMakeRange(KEY_SIZE,
KEY_SIZE)];
  NSData *iv = [self generateRandom: IV_SIZE];

  NSData *encrypted = [self aesEncrypt: payload withKey: aesKey
withIV: iv];
  NSString *basePayload = [encrypted unpaddedBase64URLEncoded];
  NSString *baseHeader = [[header
dataUsingEncoding:NSUTF8StringEncoding] unpaddedBase64URLEncoded];
  NSString *baseIV = [iv unpaddedBase64URLEncoded];
  // create auth hash
  NSData *hmac = [self hmac: encrypted withKey: hmacKey withIV: iv
withA: [baseHeader dataUsingEncoding:NSASCIIStringEncoding]];

  return [NSString stringWithFormat:@"%@..%@.%@.%@", baseHeader,
baseIV, basePayload, [[self hmacToTag: hmac]
unpaddedBase64URLEncoded]];
}
```

3.1.3 Windows 10 Mobile

Using jose-rt NuGet package:

```
public static string JweEncryptDeviceData(string dd, string
publicKeyPemBytes)
  {
     return Jwt.Encode(dd, JwaAlgorithms.RSA_OAEP,
                      JweAlgorithms.A128CBC_HS256,
                           JoseRT.Rsa.PublicKey.Load(publicKeyPemBytes));
}
```

3.2 Diffie-Hellman process

When generating data needed for AReq message (when 3DS Requestor App calls getAuthenticationRequestParameters() method in Transaction interface (see 4.4 in SDK Specification)), SDK needs to generate its ephemeral keypair for Diffie-Hellman key exchange. This is defined in the Core Specification 6.2.3.1.

3.2.1 Android

On Android this is done directly using BouncyCastle:

```
public static KeyPair generateEphemeralKeyPair() throws
NoSuchAlgorithmException, InvalidAlgorithmParameterException {
   BouncyCastleProvider provider = new BouncyCastleProvider();
   ECGenParameterSpec ecGenSpec = new ECGenParameterSpec("P-256");
   KeyPairGenerator g = KeyPairGenerator.getInstance("ECDH", provider);
   g.initialize(ecGenSpec, new SecureRandom());

   return g.generateKeyPair();
}
```

3.2.2 iOS 9

On iOS 9 this requires a bit more work again, especially when using Objective-C. GMEllipticCurveCrypto (https://github.com/ricmoo/GMEllipticCurveCrypto) can be used to generate the keypair. Generated public key is not yet ready to be included in the AReq message since it is not properly ASN.1 formatted. This is done in proof-of-concept implementation by hard coding the ASN.1 envelope around the key.

```
/*!
 * Produces the following ASN.1 structure:
 *
 * | 0:d=0  | hl=2 | l= | 89 | cons: | SEQUENCE | |
 * | 2:d=1  | hl=2 | l= | 19 | cons: | SEQUENCE | |
 * | 4:d=2  | hl=2 | l= |  7 | prim: | OBJECT   | :id-ecPublicKey |
 * | 13:d=2 | hl=2 | l= |  8 | prim: | OBJECT   | :prime256v1      |
 * | 23:d=1 | hl=2 | l= | 66 | prim: | BIT        | STRING  |
 */
+ (NSString*) publicKeyToProperASN1: (NSData *)publicKey {
  NSMutableData *asn1 = [[NSMutableData alloc]init];
  char sec256r1_header[] = {0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2A, 0x86,
0x48, 0xCE, 0x3D, 0x02, 0x01, 0x06, 0x08, 0x2A, 0x86, 0x48, 0xCE, 0x3D,
0x03, 0x01, 0x07, 0x03, 0x42, 0x00};
  [asn1 appendBytes:sec256r1_header length:sizeof(sec256r1_header)];
  [asn1 appendData: publicKey];
  NSString *encoded = [asn1 base64EncodedStringWithOptions:0];
  return encoded;
}
```

This method can then be used when generating the actual Ephemeral public key:

```
...
   ephemeralKey = [GMEllipticCurveCrypto generateKeyPairForCurve:
GMEllipticCurveSecp256r1];
   [ephemeralKey setCompressedPublicKey: FALSE];

   authRequestParams.sdkEphmeralPublicKey = [JWS
publicKeyToProperASN1:[ephemeralKey publicKey]];
...
```

3.2.3 Windows 10 Mobile

```
using Windows.Security.Cryptography.Core;
using Windows.Security.Cryptography;
  authParams = new AuthenticationRequestParameters();
  AsymmetricKeyAlgorithmProvider p =
AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.Ecdsa
Sha256);
  ephemeralKey = p.CreateKeyPairWithCurveName(EccCurveNames.NistP256);

  authParams.SDKEphmeralPublicKey =
CryptographicBuffer.EncodeToBase64String(ephemeralKey.ExportPublicKey());
```

3.3 JWS signature checking

When doing the second part of SDK - ACS secure channel setup (Core specification 6.2.3), SDK starts by validating signature of JWS structure in ARes.ACSSignedContent field. This field is passed to SDK through `ChallengeParameters.getAcsSignedContent` method call (SDK Specification 4.11.8). SDK needs to validate the signature to ensure that the data within the JWS structure has been passed unchanged from ACS to SDK. SDK has root certificate for each DS it communicates and it can use this root certificate to validate the certificate chain in JWS structure.

3.3.1 Android

On Android, JWS structure validation can be done using again Nimbus JOSE implementation. The following code sample shows how it can be done give the root certificate and the JWS structure.

```
public static String jwsValidateSignatureAndReturnBody(String jws) {
   JWSObject jwsObject;
   try {
      jwsObject = JWSObject.parse(jws);
   } catch (ParseException e) {
      throw new RuntimeException("JWS parsing failed");
   }
   try {
      JWSVerifier verifier = null;
      JWSAlgorithm alg = jwsObject.getHeader().getAlgorithm();
      if (alg.equals(JWSAlgorithm.PS256) || alg.equals(JWSAlgorithm.RS256))
{
         List<com.nimbusds.jose.util.Base64> x509CertChain =
jwsObject.getHeader().getX509CertChain();
         verifier = new RSASSAVerifier((RSAPublicKey)
X509CertUtils.parse(x509CertChain.get(0).decode()).getPublicKey());
      }
      else if (alg.equals(JWSAlgorithm.ES256)) {
         verifier = new
ECDSAVerifier(ECKey.parse(jwsObject.getHeader().getJWK().toJSONString()));
      }
      else {
         // unsupported algorithm
         throw new RuntimeException();
      }
      if (!jwsObject.verify(verifier)) {
        throw new RuntimeException("JWS validation failed.");
      }
   } catch (Exception e) {
   throw new RuntimeException("JWS validation failed.");
   }
   return jwsObject.getPayload().toString();
}
```

3.3.2 iOS 9

On iOS the situation is similar as with device data encryption, the JWS checking is implemented by using cryptography primitives from commonCrypto package, but the PS256 signature validation was implemented manually based on RFC 3447.

```
- (BOOL) verify:(NSString *)header withPayload:(NSString *)payload
withSignature:(NSData *)signature {

  NSString *clearHeader = [JWS base64Decode: header];
  NSData *headerData = [clearHeader
dataUsingEncoding:NSUTF8StringEncoding];
  NSDictionary *headerContents = [JWS JSONToDictionary:headerData];
  NSString *alg = headerContents[@"alg"];

  //validate signature
  // if alg == @“E256”
  {
    NSDictionary *jwk = headerContents[@"jwk"];
    NSData *xD = [JWS base64DecodeData:jwk[@"x"]];
    NSData *yD = [JWS base64DecodeData:jwk[@"y"]];

    NSMutableData *pk = [NSMutableData dataWithCapacity:65];
    char bytesToAppend[1] = {0x04};// Uncompressed raw key
    [pk appendBytes:bytesToAppend length:sizeof(bytesToAppend)];
    [pk appendData:xD];
    [pk appendData:yD];

    GMEllipticCurveCrypto *crypto = [GMEllipticCurveCrypto
cryptoForCurve:GMEllipticCurveSecp256r1];
    crypto.publicKey = pk;


    size_t hashBytesSize = CC_SHA256_DIGEST_LENGTH;
    unsigned char hashBytes[hashBytesSize];
    NSString *signinInput = [[NSArray arrayWithObjects: jwsHeader, payload,
nil] componentsJoinedByString:@"."];
    NSData *inputData = [signinInput
dataUsingEncoding:NSUTF8StringEncoding];
    CC_SHA256(inputData.bytes, (CC_LONG)inputData.length, hashBytes);

    if ([crypto verifySignature:signature forHash:[NSData
dataWithBytes:hashBytes length:hashBytesSize]]) {
      return TRUE;
    }

    NSLog(@"JWS failed");
    return FALSE;
  }
  // else if alg == @“PS256”
  {
  NSArray *x5cArray = (NSArray *)headerContents[@“x5c"];

  // validate the cert chain
  if ([self validateCertChain: x5cArray, dsRoot]) {
    NSLog(@"JWS cert chain invalid");
      return nil;
    }
    //load X509 certificate
    NSData *x509cert = [JWS base64DecodeData:(NSString *)x5cArray[0]];
    SecKeyRef publicKey = [JWS getX509CertPublicKey:x509cert];

  if (!publicKey) {
      NSLog(@"JWT public key invalid");
      return nil;
  }


  //implement RFC 3447 for PS256 signature validation manually
  }
```

3.3.3 Windows 10 Mobile

Using BouncyCastle and jose-rt NuGet package, signature validation:

```
public static Json GetJwsObject(string jwsData)
{
  string[] data = jwsData.Split(new Char[] { '.' });

  string h =
Encoding.UTF8.GetString(CryptographicBuffer.DecodeFromBase64String(data[0])
.ToArray());
  Json header = Utils.CreateImpl();
  header = header.Parse(h);

  string cert = header.GetNamedStringArray("x5c")[0];
  byte[] x509 = System.Convert.FromBase64String(cert);

  X509CertificateParser parser = new X509CertificateParser();
  X509Certificate x509Certificate = parser.ReadCertificate(x509);

  try
  {
   AsymmetricKeyParameter publicKey = x509Certificate.GetPublicKey();
   SubjectPublicKeyInfo info =
SubjectPublicKeyInfoFactory.CreateSubjectPublicKeyInfo(publicKey);

   AsymmetricKeyAlgorithmProvider provider =
AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.RsaPk
cs1);
   CryptographicKey key =
provider.ImportPublicKey(CryptographicBuffer.CreateFromByteArray(info.GetEn
coded()));

   string res = Jwt.Decode(jwsData, key);
   return Utils.CreateImpl().Parse(res);
  }
  catch (Exception e)
  {
   throw e;
  }
}
```

After the SDK has validated the JWS structure, it can do its second half of the Diffie-Hellman handshake defined in the Core specification 6.2.3. For this SDK has created a fresh Ephemeral keypair to be included AReq message through AuthenticationRequestParameters.getSDKEphemeralPublicKey method (SDK spec 4.12.6). Ephemeral key generation is described in the below examples.

3.3.4 Android

Android does the handshake using a Concat KDF class, as defined in Section 5.8.1 of NIST.800-56A, with the parameters given by the Core Specification.

The com.nimbusds.jose.crypto.ConcatKDF implementation was used as reference.

```
public static SecretKey generateECDHSecret(ECPublicKey pub, ECPrivateKey
priv, String sdkReferenceId) {
  try {
    SecretKey z = ECDH.deriveSharedSecret(pub, priv, null);

    ConcatKDF kdf = new ConcatKDF("SHA-256");
    return kdf.deriveKey(
      z,
      256,
      ConcatKDF.encodeStringData(null),
      ConcatKDF.encodeDataWithLength((Base64URL)null),
      ConcatKDF.encodeDataWithLength(Base64URL.encode(sdkReferenceId)),
      ConcatKDF.encodeIntData(256),
      ConcatKDF.encodeNoData());
    } catch (Exception e1) {
      l.error("failed to decrypt", e1);
      throw new RuntimeException();
    }
}
```

3.3.5 iOS 9

On iOS, this is done using manually implementing RFC 7518 and NIST.800-56A.

3.3.6 Windows 10 Mobile

Using BouncyCastle:

```
IBuffer acsKey =
CryptographicBuffer.DecodeFromBase64String(jws.GetNamedString(ProtocolConst
ants.ACSEphemeralPublicKey));
secretKey = Crypto.CreateSessionKey(ephemeralKey.Export(), acsKey,
sdkReferenceNumber);

public static byte[] CreateSessionKey(IBuffer privateKey, IBuffer
acsPublicKey, string sdkReferenceId)
{
  AsymmetricKeyParameter publicKeyParams =
            PublicKeyFactory.CreateKey(acsPublicKey.ToArray());
  AsymmetricKeyParameter privateKeyParams =
           PrivateKeyFactory.CreateKey(privateKey.ToArray());
  IBasicAgreement agreement =
           AgreementUtilities.GetBasicAgreement("ECDH");
  agreement.Init(privateKeyParams);

  BigInteger key = agreement.CalculateAgreement(publicKeyParams);

  // pad with leading zeroes if needed
  byte[] k = key.ToByteArrayUnsigned();

  byte[] buffer = Enumerable.Repeat((byte)0x00, 32).ToArray();
  Array.ConstrainedCopy(k, 0, buffer, (buffer.Length - k.Length),
                        k.Length);

  DataWriter writer = new DataWriter();
  writer.WriteBytes(buffer);
  IBuffer z = writer.DetachBuffer();
  writer.Dispose();

  ConcatKDF kdf = new ConcatKDF(HashAlgorithmNames.Sha256);

  return kdf.DeriveKey(JoseRT.Util.Buffer.ToBytes(z),
         256,
         ConcatKDF.EncodeStringData(null),
         ConcatKDF.EncodeDataWithLength((string)null),
         ConcatKDF.EncodeDataWithLength(Base64Url.Encode(
                      Encoding.UTF8.GetBytes(sdkReferenceId))),
         ConcatKDF.EncodeIntData(256),
         ConcatKDF.EncodeNoData());
}
```

3.4 Encryption of CReq / Decryption of CRes

Once the secret key is generated in SDK, it can be used to encrypt CReq and decrypt CRes messages. In case of A128CBC-HS256 only one key is used for both directions (i.e. for both CReq encryption and for CRes decryption) whereas with A128GCM both directions have separate keys.

3.4.1 Android

As with other encryption operations, this is also done with Nimbus JOSE.

```
public byte[] encrypt(JSONObject challengeRequest) {
  try {
    final byte counter = sdkCounterStoA;
    JWEHeader header = new JWEHeader.Builder(JWEAlgorithm.DIR,
EncryptionMethod.A128GCM)
       .keyID((String)
challengeRequest.get(ProtocolConstants.ACSTransactionID))
         .build();
    challengeRequest.put(ProtocolConstants.SDKCounterStoA,
String.format("%03d",counter));

    JWEObject jweObject = new JWEObject(header, new
Payload(challengeRequest.toString()));
```

```
     jweObject.encrypt(new
TransactionEncrypter(Arrays.copyOfRange(secretKey.getEncoded(),0,16),
counter));

     byte[] bytes = jweObject.serialize().getBytes();

     sdkCounterStoA ++;
     if (sdkCounterStoA == 0) {
       throw new RuntimeException("SdkCounterStoA zero");
     }

     return bytes;

     } catch (Exception e) {
       log.info("JWE failed");
       return challengeRequest.toString().getBytes();
     }
}
```

The reference implementation has implemented the TransactionEncrypter-class as follows:

```
public final class TransactionEncrypter extends
com.nimbusds.jose.crypto.DirectEncrypter{

    private byte counter;

    public TransactionEncrypter(byte[] key, byte counter) throws
KeyLengthException {
      super(new SecretKeySpec(key, "AES"));
      this.counter = counter;
    }

    @Override
    public JWECryptoParts encrypt(JWEHeader header, byte[] clearText)
throws JOSEException {
      JWEAlgorithm alg = header.getAlgorithm();

      if (! alg.equals(JWEAlgorithm.DIR)) {
       throw new JOSEException("Invalid alg " + alg);
      }

      // Check key length matches encryption method
      EncryptionMethod enc = header.getEncryptionMethod();

      if (enc.cekBitLength() != ByteUtils.bitLength(getKey().getEncoded()))
{
       throw new KeyLengthException(enc.cekBitLength(), enc);
      }

      final Base64URL encryptedKey = null; // The second JWE part

      if (enc.cekBitLength() != ByteUtils.bitLength(getKey().getEncoded()))
{
       throw new KeyLengthException("The Content Encryption Key (CEK)
length for " + enc + " must be " + enc.cekBitLength() + " bits");
      }

      // Apply compression if instructed
      byte[] plainText = DeflateHelper.applyCompression(header, clearText);

      // Compose the AAD
      byte[] aad = AAD.compute(header);

      // Encrypt the plain text according to the JWE enc
      byte[] iv = Crypto.getGcmIvStoA(counter);
      AuthenticatedCipherText authCipherText;

      if
(header.getEncryptionMethod().equals(EncryptionMethod.A128CBC_HS256)) {
       authCipherText = AESCBC.encryptAuthenticated(
          getKey(), iv, plainText, aad,
          getJCAContext().getContentEncryptionProvider(),
          getJCAContext().getMACProvider());

      } else if
(header.getEncryptionMethod().equals(EncryptionMethod.A128GCM)) {
       authCipherText = AESGCM.encrypt(
          getKey(), iv, plainText, aad,
          getJCAContext().getContentEncryptionProvider());

      } else {
       throw new
JOSEException(AlgorithmSupportMessage.unsupportedEncryptionMethod(
          header.getEncryptionMethod(),
          SUPPORTED_ENCRYPTION_METHODS));
      }

      return new JWECryptoParts(
         header,
         encryptedKey,
         Base64URL.encode(iv),
         Base64URL.encode(authCipherText.getCipherText()),
         Base64URL.encode(authCipherText.getAuthenticationTag()));
    }
}
```

Below is an example how to implement Crypto.getGcmIvStoA()

```
  public static byte[] getGcmIvStoA(byte sdkCounterStoA) {
    return getGcmId((byte)0x00, sdkCounterStoA);
  }

  public static byte[] getGcmIvAtoS(byte sdkCounterAtoS) {
    return getGcmId((byte)0xFF, sdkCounterAtoS);
  }

  private static byte[] getGcmId(byte pad, byte counter) {
    byte[] iv = new byte[16];
    Arrays.fill(iv, pad);
    iv[iv.length-1] = counter;
    return iv;
  }
```

Decryption is also done with Nimbus JOSE.

```
try {
  byte[] key = secretKey.getEncoded();
  JWEObject jweObject = JWEObject.parse(message);

  EncryptionMethod jweMethod =
jweObject.getHeader().getEncryptionMethod();
  if (jweMethod == EncryptionMethod.A128GCM) {
key = Arrays.copyOfRange(key, key.length-16, key.length);
  }

  jweObject.decrypt(new DirectDecrypter(key));

  JSONObject challengeResponse = new
JSONObject(jweObject.getPayload().toString());

  byte acsCounterAtoS =
Byte.valueOf(challengeResponse.getString(ProtocolConstants.ACSCounterAtoS))
;
  if (sdkCounterAtoS != acsCounterAtoS) {
throw new RuntimeException("counters (" + sdkCounterAtoS + "/" +
acsCounterAtoS + ")");
  }

  sdkCounterAtoS ++;
  if (sdkCounterAtoS == 0) {
throw new RuntimeException("SDKCounterAtoS zero");
  }

  return challengeResponse;
} catch (Exception e) {
  log.error("failed to decrypt", e);
  throw new RuntimeException();
}
```

3.4.2 iOS 9

On iOS this operation is again implemented manually, using crypto primitives from commonCrypto and then crafting the JSON structure by hand. One thing to note here is that A128CBC-HS256 seemed to be the only one of the algorithms mentioned in the Core Specification 6.2.3 that worked at the time of writing.

```
+(NSString *) createJWE: (NSData *)payload withSecret: (NSData *)secret
withKid: (NSString *)kid {
  NSData *hmacKey  = [secret subdataWithRange:NSMakeRange(0, KEY_SIZE)];
  NSData *aesKey = [secret subdataWithRange:NSMakeRange(KEY_SIZE,
KEY_SIZE)];

  NSData *iv = [self generateRandom: IV_SIZE];
  // create header
  NSString *header = [NSString stringWithFormat:
@"{\"kid\":\"%@\",\"enc\":\"A128CBC-HS256\",\"alg\":\"dir\"}", kid];
  // encrypt payload
  NSData *encrypted = [self aesEncrypt: payload withKey: aesKey withIV:
iv];
  NSString *basePayload = [encrypted unpaddedBase64URLEncoded];
  NSString *baseHeader = [[header dataUsingEncoding:NSUTF8StringEncoding]
unpaddedBase64URLEncoded];
  NSString *baseIV = [iv unpaddedBase64URLEncoded];
  // create auth hash
  NSData *hmac = [self hmac: encrypted withKey: hmacKey withIV: iv withA: 
[baseHeader dataUsingEncoding:NSASCIIStringEncoding]];

  return [NSString stringWithFormat:@"%@..%@.%@.%@", baseHeader, baseIV, 
basePayload, [[self hmacToTag: hmac] unpaddedBase64URLEncoded]];
}
```

After this function is called, the following checking is done:

```
  sdkCounterStoA++;
  if (sdkCounterStoA == 0) {
   NSDictionary *errorDictionary = @{ NSLocalizedDescriptionKey: 
@"SdkCounterStoA zero"};
   NSError *error = [[NSError alloc] initWithDomain:@"MI_SDK" code:-1 
userInfo:errorDictionary];
   @throw error;
  }
```

Decrypting example below:

```
+(NSString *)parseJWE: (NSString *) based withSecret: (NSData *)secret {

  // base64 url encoding is not exactly base64...
  // https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-
41#appendix-C
  //
  //NSString *based = [[jwe stringByReplacingOccurrencesOfString:@"_" 
withString: @"/"] stringByReplacingOccurrencesOfString:@"-" 
withString:@"+"];

  //NSData *decodedSecret = [self base64DecodeData:secret];
  NSData *hmacKey  = [secret subdataWithRange:NSMakeRange(0, KEY_SIZE)];
  NSData *aesKey = [secret subdataWithRange:NSMakeRange(KEY_SIZE, 
KEY_SIZE)];

  NSArray *components = [based componentsSeparatedByString:@"."];

  /* this is useful when debugging...
  [components enumerateObjectsUsingBlock:^(id object, NSUInteger idx, BOOL 
*stop) {
   NSLog(@"%lu: %@", (unsigned long)idx, [object description]);
  }]; */

  // JWE contains 5 components
  // 1 Header
  // we need this to do the hmac
  NSString *header = [components objectAtIndex:0];

  // we only support one algorithm, and we don't really need kid on this 
side
  // so this can be left here waiting for some use
  //NSString *clearHeader = [self base64Decode: header];

  // 2 encrypted key
  // we don't use key, and I hate warnings, so it is commented for now
  //NSString *key = [components objectAtIndex:1];
  // 3 IV
  NSString *iv = [components objectAtIndex:2];
  NSData *decodedIV = [self base64DecodeData:iv];

  // 4 payload
  NSString *payload = [components objectAtIndex:3];
  NSData *decodedPayload = [self base64DecodeData: payload];

  // 5 Tag
  NSString *tag = [components objectAtIndex:4];

  NSData *decrypted = [self aesDecrypt: decodedPayload withKey: aesKey
withIV: decodedIV];
  NSString* decryptedPayload = [[NSString alloc] initWithData:decrypted
encoding:NSUTF8StringEncoding];
  NSData *hmac = [self hmac: decodedPayload withKey: hmacKey withIV:
decodedIV withA: [header dataUsingEncoding:NSASCIIStringEncoding]];
  NSString *calculatedTag = [[self hmacToTag: hmac]
unpaddedBase64URLEncoded];
  decryptedPayload = [decryptedPayload stringByTrimmingCharactersInSet:
[NSCharacterSet whitespaceAndNewlineCharacterSet]];
  NSRange r = [decryptedPayload rangeOfString:@"}"
options:NSBackwardsSearch];
  decryptedPayload = [decryptedPayload substringWithRange: NSMakeRange(0,
r.location + 1)];
  if (! [tag isEqualToString: calculatedTag]) {
   NSLog(@"TAG doesn't match: message = %@, calculated = %@", tag,
calculatedTag);
   return decryptedPayload;
  }
  return decryptedPayload;
}
```

After this function is called, the following checking is done:

```
  NSError *serializationError = nil;
  NSLog(@"data2: %@", [[NSString alloc] initWithData:decrypted
encoding:NSUTF8StringEncoding]);
  id json = [NSJSONSerialization JSONObjectWithData:decrypted options:0
error:&serializationError];
  if (serializationError) {
   SDKLogE(@"Serialization error %@",serializationError);
   @throw serializationError;
  }

  uint8_t acsCounterAtoS = [json[KACSCounterAtoS] intValue];
  if (sdkCounterAtoS != acsCounterAtoS)
  {
   SDKLogE(@"counters not equal");
   return nil;
  }

  sdkCounterAtoS++;
  if (sdkCounterAtoS == 0) {
   NSDictionary *errorDictionary = @{ NSLocalizedDescriptionKey:
@"SdkCounterAtoS zero"};
```

```
  NSError *error = [[NSError alloc] initWithDomain:@"MI_SDK" code:-1
userInfo:errorDictionary];
  @throw error;
}
```

3.4.3 Windows 10 Mobile

Using jose-rt NuGet package:

```
public Json Decrypt(string message)
{
  Part[] jweParts = Compact.Parse(message);

  Json jweHeader =
Utils.CreateImpl().Parse(Encoding.UTF8.GetString(jweParts[0].Bytes));
  string decrypted;

  if
(jweHeader.GetNamedString("enc").Equals(JoseRT.JweAlgorithms.A128GCM))
  {
decrypted = Jwt.Decode(message, Arrays.FirstHalf(secretKey));
  } else
  {
decrypted = Jwt.Decode(message, secretKey);
  }

  Json challengeResponse = Utils.CreateImpl().Parse(decrypted);
  byte acsCounterAtoS =
Convert.ToByte(challengeResponse.GetNamedString(ProtocolConstants.ACSCounte
rAtoS),16);
  if (sdkCounterAtoS != acsCounterAtoS)
  {
throw new Exception("counters (" + sdkCounterAtoS + "/" + acsCounterAtoS +
")");
  }

  sdkCounterAtoS++;
  if (sdkCounterAtoS == 0)
  {
throw new Exception("counter zero");
  }

  return challengeResponse;
}


public string Encrypt(Json challengeRequest)
{
  byte counter = sdkCounterStoA;
  DirectKeyManagement algorithm = new DirectKeyManagement();

  var jwtHeader = new JsonObject {
  {"enc",JsonValue.CreateStringValue(JoseRT.JweAlgorithms.A128CBC_HS256)
},
  {"alg", JsonValue.CreateStringValue(JoseRT.JwaAlgorithms.DIR) },
  {"kid",
JsonValue.CreateStringValue(challengeRequest.GetNamedString(ProtocolConstan
ts.ACSTransactionID)) }
};
```

```
   challengeRequest.Add(ProtocolConstants.SDKCounterStoA,
Convert.ToString(counter));

   Part[] keys = algorithm.WrapNewKey(256, secretKey, jwtHeader);
   Part cek = keys[0];
   Part encryptedCek = keys[1];
   Part header = Part.New(jwtHeader.Stringify());

   byte[] plainText = Encoding.UTF8.GetBytes(challengeRequest.Stringify());

   byte[] aad = Encoding.UTF8.GetBytes(Compact.Serialize(header));
   AesCbcHmacEncryptor encryptor = new AesCbcHmacEncryptor(256);
   Part[] encParts = encryptor.Encrypt(aad, plainText, cek.Bytes);

   string encrypted = Compact.Serialize(header, encryptedCek, encParts[0],
encParts[1], encParts[2]);

   sdkCounterStoA++;
   if (sdkCounterStoA == 0)
   {
throw new Exception("counter zero");
   }

   return encrypted;
}
```

3.5 Implementation of Security Features

The SDK implements a number of security functions that are defined in more detail in 3DS SDK specification and in PCI 3DS Security Standard. Payment Systems may also have additional requirements related to SDK security features.

During SDK initialization, the SDK performs a number of security checks that are defined in Chapter 8.2 of the SDK Specification. Possible security warnings are made available to the 3DS Requestor Application. In the following chapters (3.5.1 – 3.5.5), implementation examples are given of each security check.

Table 2 3DS SDK Initialization Security Checks

Security Warning ID	Description	Severity Level
SW01	The device is jailbroken.	HIGH
SW02	The integrity of the SDK has been tampered.	HIGH
SW03	An emulator is being used to run the App.	HIGH
SW04	A debugger is attached to the App.	MEDIUM

Security Warning ID	Description	Severity Level
SW05	The OS or the OS version is not supported.	HIGH

In addition to the security checks listed above, this chapter gives implementation examples of further security features (chapters 3.5.6 – 3.5.15).

The examples presented in this chapter do not fully cover the requirements but rather give guidance on how the requirements can be fulfilled. The code samples should not be used directly in production code.

3.5.1 SW01: The device is jailbroken

This check shall be conducted during initialization and make the result of the checks available as a list of warnings to the 3DS Requestor App.

Android

On android the typical and simple way to check if root access is enabled is to look for typical root binaries such as su, and sudo binaries from binary paths, and typical root apps, such as superuser.apk.

Below is an example of a possible implementation:

```
public static final List<String> BINARY_PATHS = Arrays.asList("/sbin/",
"/system/bin/", "/system/xbin/", "/data/local/xbin/",
"/data/local/bin/", "/system/sd/xbin/", "/system/bin/failsafe/",
"/data/local/");

public static final List<String> APK_PATHS = Arrays.asList("/system/app/");

public static String isRooted() {
  return findSuBinary() + findSuperuserAPK();
}

public static String findSuBinary() {
  String s = findFileInPaths("su", BINARY_PATHS);
  if (s != null && s.length() > 0) {
return "su binary was found on paths: " + s;
  }
  return "";
}

public static String findSuperuserAPK() {
  String s = findAPK("Superuser");
  if (s != null && s.length() > 0) {
return "Superuser apk was found on paths: " + s;
  }
  return "";
}
```

iOS

On iOS jailbreaking can be detected multiple ways. Some of the more popular options are:

- detecting files created during jailbreaking process (for example /private/var/tmp/cydia.log)
- detecting symlinks of small partitions (such as /usr/include)

- checking API calls, for instance system() return values
- detecting whether cydia:// scheme is available

Windows 10 Mobile

There is currently no way to determine if Developer Mode is turned on in the device, but "Windows.ApplicationModel.Package.Current.IsDevelopmentMode" can be used to detect if the app was installed in development mode.

More information on the topic can be found on the Microsoft Developer site at:

https://msdn.microsoft.com/library/windows/apps/dn175745

3.5.2 SW02: The integrity of the SDK has been tampered

This check shall be conducted during initialization and make the result of the checks available as a list of warnings to the 3DS Requestor App.

Android

In Android, the checking of the SDK integrity can only be accomplished by implementing class consistency verification using standard Java language reflection (java.lang.reflect.*).

The SDK .aar Android Java library release binary cannot be signed because you can only sign the actual application release binary.

iOS

In iOS, the checking of the SDK integrity can only be accomplished by implementing class consistency verification using objective-c/swift language reflection (for example NSClassFromString(NSString *className)) etc.

You cannot sign the SDK framework release binary that is referenced by the actual application because you can only sign the actual application release binary.

Windows 10 Mobile

In Windows 10 Mobile, the checking of the SDK integrity can only be accomplished by producing class consistency verification using UWP (Universal Windows Platform) C# language reflection.

Below is a link for more information. The SDK portable class library binary that is referenced by the actual application cannot be signed because you can only sign the actual application release binary.

https://msdn.microsoft.com/en-us/library/system.reflection.typeinfo.declaredproperties(v=vs.110).aspx

3.5.3 SW03: An emulator is being used to run the App

This check shall be conducted during initialization and make the result of the checks available as a list of warnings to the 3DS Requestor App.

Android

On android easiest way to check if code is running in emulator is to check for build finger print:

```
public static boolean isRunningInEmulator() {
   return Build.FINGERPRINT.startsWith("generic")
   || Build.FINGERPRINT.startsWith("unknown")
   || Build.MODEL.contains("Emulator")
   || Build.MODEL.contains("Android SDK built for x86")
   || Build.MODEL.contains("google_sdk")
   || Build.MANUFACTURER.contains("Genymotion")
   || (Build.BRAND.startsWith("generic") &&
Build.DEVICE.startsWith("generic"))
   || "google_sdk".equals(Build.PRODUCT);
}
```

iOS

To detect if SDK is running in simulator in iOS, SDK can include conditional code that is executed when running in Intel platforms:

```
#if (arch(i386) || arch(x86_64))
// return warning here
#endif
```

Windows 10 Mobile

The device model can be checked to determine whether the Windows 10 mobile emulator is being used. Below is an example of an implementation:

```
EasClientDeviceInformation eas = new EasClientDeviceInformation();
string c02DeviceModel = eas.SystemManufacturer + " " +
eas.SystemProductName;

returns: "Microsoft Virtual" for emulator
```

3.5.4 SW04: A debugger is attached to the App

This check shall be conducted during initialization and make the result of the checks available as a list of warnings to the 3DS Requestor App.

Android

On android emulator usage can be detected by calling android.os.Debug.isDebuggerConnected() method.

```
if (android.os.Debug.isDebuggerConnected()) {
   warnings.add(new Warning(Warning.SE02_DEBUGGER, "Running in debugger",
Warning.Severity.MEDIUM));
 }
```

iOS

According to Apple documentation the code below is the way to check if code is running under debugger.

For more information on this, see the documentation on Apple developer site: https://developer.apple.com/library/content/qa/qa1361/_index.html

```
#include <assert.h>
#include <stdbool.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/sysctl.h>
```

```
static bool AmIBeingDebugged(void)
  // Returns true if the current process is being debugged (either
  // running under the debugger or has a debugger attached post facto).
{
  int junk;
  int mib[4];
  struct kinfo_proc   info;
  size_t   size;

  // Initialize the flags so that, if sysctl fails for some bizarre
  // reason, we get a predictable result.

  info.kp_proc.p_flag = 0;

  // Initialize mib, which tells sysctl the info we want, in this case
  // we're looking for information about a specific process ID.

  mib[0] = CTL_KERN;
  mib[1] = KERN_PROC;
  mib[2] = KERN_PROC_PID;
  mib[3] = getpid();

  // Call sysctl.

  size = sizeof(info);
  junk = sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, NULL, 0);
  assert(junk == 0);

  // We're being debugged if the P_TRACED flag is set.

  return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
}
```

Windows 10 Mobile

On Windows 10 mobile, the following system function can be called to detect whether the debugger is attached:

```
using System.Diagnostics;
if (Debugger.IsAttached) {
...
}
```

3.5.5 SW05: The OS or the OS version is not supported

This check shall be conducted during initialization and make the result of the checks available as a list of warnings to the 3DS Requestor App.

Android

Below are two examples of how the OS version check can be implemented in Android:

Example 1:

```
String c02DeviceModel = Build.MANUFACTURER + " " + Build.MODEL;
String c03OSName = getOSName();
String c04OSVersion = Build.VERSION.RELEASE;

private String getOSName() {
  Field[] fields = Build.VERSION_CODES.class.getFields();
  for (Field field : fields) {
   String fieldName = field.getName();
   int fieldValue = -1;

   try {
     fieldValue = field.getInt(new Object());
   } catch (Exception e) {
   }
   if (fieldValue == Build.VERSION.SDK_INT) {
     return fieldName;
   }
  }
  return PLATFORM_DOES_NOT_SUPPORT_OR_DEPRECATED;
}
```

Example 2:

```
public static boolean isAndroid_5_0_orNewer() {
   boolean value = (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP);
   log.debug("isAndroid_5_0_orNewer: " + value);
   return value;
}
```

iOS

Below is listed how to implement OS verion checking in iOS.

```
UIDevice *currentDevice = [UIDevice currentDevice];
NSString* c03OSName = [currentDevice systemName];
NSString* c04OSVersion = [currentDevice systemVersion];
```

Another way to detect the OS version is listed below:

```
+ (BOOL) isIOS9OrNewer {
  if ([NSProcessInfo
instancesRespondToSelector:@selector(isOperatingSystemAtLeastVersion:)]) {
   NSOperatingSystemVersion ios9_0_0 = (NSOperatingSystemVersion){9, 0, 0};
   if ([[NSProcessInfo processInfo]
isOperatingSystemAtLeastVersion:ios9_0_0]) {
     // iOS 9.0
     return YES;
   } else {
     // Older than iOS 9.0
     return NO;
   }
  } else {
   // Older than iOS 8.0
   return NO;
  }
}
```

Windows 10 Mobile

Here is an example of implementation in the Windows platform to detect the OS version:

```
using Windows.ApplicationModel;
using Windows.Security.ExchangeActiveSyncProvisioning;
using Windows.System.Profile;
```

```
...
public static class Info
{
  public static string SystemFamily { get; }
  public static string SystemVersion { get; }
  public static string SystemArchitecture { get; }
  public static string ApplicationName { get; }
  public static string ApplicationVersion { get; }
  public static string DeviceManufacturer { get; }
  public static string DeviceModel { get; }

  static Info()
  {
// get the system family name
AnalyticsVersionInfo ai = AnalyticsInfo.VersionInfo;
SystemFamily = ai.DeviceFamily;

// get the system version number
string sv = AnalyticsInfo.VersionInfo.DeviceFamilyVersion;
ulong v = ulong.Parse(sv);
ulong v1 = (v & 0xFFFF000000000000L) >> 48;
ulong v2 = (v & 0x0000FFFF00000000L) >> 32;
ulong v3 = (v & 0x00000000FFFF0000L) >> 16;
ulong v4 = (v & 0x000000000000FFFFL);
SystemVersion = $"{v1}.{v2}.{v3}.{v4}";

// get the package architecure
Package package = Package.Current;
SystemArchitecture = package.Id.Architecture.ToString();

// get the user friendly app name
ApplicationName = package.DisplayName;

// get the app version
PackageVersion pv = package.Id.Version;
ApplicationVersion = $"{pv.Major}.{pv.Minor}.{pv.Build}.{pv.Revision}";

// get the device manufacturer and model name
EasClientDeviceInformation eas = new EasClientDeviceInformation();
DeviceManufacturer = eas.SystemManufacturer;
DeviceModel = eas.SystemProductName;
  }
}
```

The example above would return

```
Windows.Mobile
10.0.10166.0
Arm
Info Test
1.1.0.0
NOKIA
RM-940_nam_att_200
```

For more information on the topic:

https://www.suchan.cz/2015/08/uwp-quick-tip-getting-device-os-and-app-info/

3.5.6 User data in Native and WebView UI

Android

Android platform has some limitations when accessing form input data programmatically. Android allows accessing GET parameters, but not POST parameters.

In Native UI, the OTP should be queried using a subclass. Below is an example of a possible implementation.

```
class SDKEditText extends EditText {
  SDKEditText(Context ctx) {
super(ctx);
  }

  @Override
  public Editable getText() {
return new SpannableStringBuilder();
  }

  protected CharSequence getTextInternal() {
if(Looper.myLooper() == Looper.getMainLooper()) {
  return super.getText();
}
return null;
  }
}
```

For the HTML WebView, the example below is one possible implementation, where the WebView class is extended so that JavaScript, content access and caching are disabled.

```
class SDKWebView extends WebView {
  public SDKWebView(Context context) {
this(context, null);
  }

  public SDKWebView(Context context, AttributeSet attributeSet) {
super(context, attributeSet);
super.getSettings().setJavaScriptEnabled(false);
super.getSettings().setAllowContentAccess(false);
super.getSettings().setCacheMode(WebSettings.LOAD_NO_CACHE);
  }

  @Override
  public WebSettings getSettings() {
return null;
  }

  @Override
  public void addJavascriptInterface(Object object, String name) {}

  @Override
  public void setWebViewClient(WebViewClient client) {}

  protected void setWebViewClientInternal(WebViewClient client) {
super.setWebViewClient(client);
  }

}
```

Override the following methods of the WebViewClient class:

1. shouldOverrideUrlLoading

This method gives the host application a chance to take over the control when a new URL is about to be loaded in the current WebView. At this point, determine if the URL is allowed to be loaded. If not, then it should be blocked.

2. onLoadResource

This method notifies the host application that the WebView will load the resource specified by the given URL. At this point, intercept any external resource loading in WebView.

```
private class SDKWebViewClient extends WebViewClient {
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
  if (url.compareToIgnoreCase("https://EMV3DS/challenge") == 0) {
return super.shouldOverrideUrlLoading(view, url);
  }
  return false;
}

@Override
public void onLoadResource(WebView view, String url) {
  }
}
```

iOS

The following sections contain information about the approach that a security review provided regarding securing the 3DS SDK UI for iOS.

WebView UI on iOS

In the HTML UI format, the ACS is responsible for providing the content that is displayed to the Cardholder during the Challenge Flow. This content is provided as a fully-formed HTML snippet. The ACS encrypts the HTML and sends it to the 3DS SDK in the CRes message.

The 3DS SDK shall decrypt the HTML and use a web view to display the content on the mobile device. The Cardholder data shall then be captured and sent to the ACS in the CReq message.

To make it difficult for rogue apps to intercept the messages, the following approach was used:

- Use the WKWebView class for displaying WebViews.
- Do not directly instantiate the WKWebView.
- Define a new class to sub-class the WKWebView class.
- Override the following in the newly defined sub-class:
    - Property configuration
        - Override this property from the WKWebView class to return a new empty WKWebViewConfiguration object. This ensures that the actual configuration object is not available outside the subclass for unwanted manipulations.
        - Make the property read-only by providing an empty implementation in the setter to prevent unwarranted external manipulation.

- Method evaluateJavaScript
  - Override this method from the WKWebView class with an empty implementation. This ensures that no arbitrary JavaScripts externally injected are executed by the 3DS SDK.

Native UI on iOS

Native UI is designed to seamlessly integrate with the Merchant App. Native UI uses native components, such as buttons or list boxes, which allows for a look and feel that is consistent with the Merchant App's checkout experience. The 3DS SDK shall create the UI elements and control the rendering of the UI.

It is important to secure UI elements that capture sensitive Cardholder data. For example, a text field that is used to capture the Cardholder’s password or secure code must be secured.

The following approach was used to secure a text box field:

- Do not directly use the UITextField in the storyboard.
- Define a new custom class to sub-class the UITextField class.
- Mark the class final and internal.
- Override the following in the newly defined sub-class:
  - Property text
    - Override this property from the UITextField class to return nil in the getter. This will ensure that the actual text is not available outside the subclass for unwanted manipulations.
    - Make the property read-only by providing an empty implementation in the setter to prevent unwarranted external manipulation.
    - Set the access level of the property to internal.
  - Property delegate
    - Override this property from the UITextField class to return nil in the getter.
    - Make the property read-only by providing an empty implementation in the setter to prevent unwarranted external manipulation. This ensures that "delegates" cannot be manipulated from external modules to prevent unauthorized access to text field messages.
    - Set the access level of the property to internal.
- Define the following in the newly defined sub-class:
  - Property textInternal
    - getter will return the actual text value from the text property of the UITextField by calling super.text.setter will save the new value in the text property of the UITextField by calling super.text Set the access level of the property to internal.
  - Property delegateInternal
    - getter will return the actual delegate value from the delegate property of the UITextField by calling super.delegate.setter will save the new value in the "delegate" property of the UITextField by calling super.delegate. Set the access level of the property to internal.

3.5.7 External URL requests in the HTML UI WebView

Android

A private intercept function is used in the WebChallengeActivity to intercept all WebView requests to either process the request locally or to ignore the request.

```
private WebResourceResponse intercept(Uri requestUri) {
  String uri = requestUri.toString();
  boolean intercept = uri.startsWith(ProtocolConstants.HtmlVerify);

  if (intercept) {
String challengeCode = requestUri.getQueryParameter(CHALLENGE_PARAMETER);
String submit = requestUri.getQueryParameter(ACTION_PARAMETER);

if (submit == null) {
  log.error("No action found");
}
else {
  if (submit.equalsIgnoreCase(ACTION_VERIFY) && challengeCode != null) {
challengeCodeEntered(challengeCode);
  } else if (submit.equalsIgnoreCase(ACTION_RESEND)) {
challengeResendCode();
  } else {
log.error("Unknown action[" + submit + "/"+ challengeCode + "]");
  }
}
return new WebResourceResponse("text/html", "UTF-8", new
ByteArrayInputStream(new byte[0]));
  }
  return null;
}
```

iOS

The UIWebView shouldStartLoadWithRequest method is overridden to intercept all HTML UI requests for local processing.

```
(BOOL)webView:(UIWebView *)webView
  shouldStartLoadWithRequest:(NSURLRequest*)request
  navigationType:(UIWebViewNavigationType)navigationType
```

Windows 10 Mobile

NavigationStarting

This event occurs before the WebView navigates to new content. At this point, determine if the URL is allowed to be loaded, i.e. HTTPS://EMV3DS/challenge. If not, then it should be blocked.

For more information about the topic, see the Microsoft Developer Network for more details.

https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.webview.navigationstarting.aspx

ContentLoading

This event occurs when the WebView has started loading new content. At this point, intercept any external resource loading in WebView.

For more information about the topic, see the Microsoft Developer Netowork for more details.

https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.webview.contentloading.aspx

3.5.8 Injection and execution of JavaScript in the HTML UI

Android

An SDKWebView class which extends WebView is used to disable JavaScript code in the HTML UI and to prevent external packages from overriding the WebViewClient.

```
public SDKWebView(Context context, AttributeSet attributeSet) {
  super(context, attributeSet);
  super.getSettings().setJavaScriptEnabled(false);
  super.getSettings().setAllowContentAccess(false);
  super.getSettings().setCacheMode(WebSettings.LOAD_NO_CACHE);
}

@Override
public WebSettings getSettings() {
  return null;
}

@Override
public void addJavascriptInterface(Object object, String name) {
}

@Override
public void setWebViewClient(WebViewClient client) {
}

protected void setWebViewClientInternal(WebViewClient client) {
  super.setWebViewClient(client);
}
```

iOS

While the iOS UIWebView does not yet allow disabling of JavaScript, the WKWebView class introduced for iOS 8 and above allows disabling of JavaScript via the WKPreferences property javaScriptEnabled.

More information on the topic can be found at:
https://developer.apple.com/reference/uikit/uiwebview

Windows 10 Mobile

In Windows 10 mobile, the WebView.Settings.IsJavaScriptEnabled can be set to false to disable JavaScript in the WebView.

The following links provide more information on the topic:

https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.webview.settings.aspx

https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.webviewsettings.aspx

3.5.9 Protection from reverse engineering

Android

Obfuscation on Android platform is typically done with either included ProGuard, or with one of the commercial obfuscators such as DexGuard. Important thing to note is that while obfuscation can make code harder for anyone to read or reason about, runtime environment still needs to have access to all resources and code, and thus the obfuscation is just making things harder for potential attacker, not preventing access to any resource or code.

Here is a sample configuration for SDK obfuscation with ProGuard. Please note that it is better to obfuscate the whole app, because otherwise the whole SDK interface needs to be left unobfuscated.

In the build.gradle of the SDK, the following configuration will enable proguard obfuscation:

```
buildTypes {
  release {
minifyEnabled true
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
  }
  debug {
minifyEnabled false
  }
}
```

The default proguard-android.txt configuration is normally sufficient for the project: https://android.googlesource.com/platform/sdk/+/master/files/proguard-android.txt

In proguard-rules.pro, explicitly keep the classes that should be visible to the application, exposing only the public methods, for example:

```
-keep class ThreeDSecurev2Service {public *;}
-keep class UiCustomization {public *;}
-keep class ButtonCustomization {public *;}
```

For these classes, it has to be ensured that only the needed methods are public, and if possible set the class to final to prevent it from being extended.

Note that the class names of activities included in the AndroidManifest.xml are automatically kept to allow launching by the Android system.

iOS

For iOS there is no standard obfuscation tool. There are some commercial tools such as Metaforic and Arxan, and some open source tools such as obfuscator-llvm and iOS-Class-Guard.

For more information on the tools, more information is available on their web sites.

3.5.10 Disabling screenshots

Android

Screenshots can be disabled in a straightforward way. An example of this is below:

```
protected void disableScreenshotsImpl() {
   if (!Settings.getInstance().isScreenshotsAllowed()) {
getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
WindowManager.LayoutParams.FLAG_SECURE);
   }
}
```

iOS

On iOS 9 there is a restriction in configuration profile that can be used to prevent screenshots of the application.

For more information on this, see the documentation on Apple developer site: https://developer.apple.com/library/content/featuredarticles/iPhoneConfigurationProfileRef/Introduction/Introduction.html

Windows 10 Mobile

Screenshots may be disabled with the following code:

```
if (!Settings.GetInstance().IsScreenshotsAllowed())
{
  log.Debug("Disabling screenshots [" + this + "]");
  Windows.UI.ViewManagement
     .ApplicationView.GetForCurrentView().IsScreenCaptureEnabled = false;
}
```

3.5.11 Run-time integrity checks

There are many ways to implement the SDK to comply with this requirement.

- Obfuscation is one way to prevent modifications of the SDK.
  See chapter 3.5.9 Protection from reverse engineering for more information
- On platforms that use native code, forced inlining can also be used to protect from functionality being modified
- Calculating check sums over assembly code can be used to keep track of the codebase and check for integrity. For more details on the topic, see the following article: https://applidium.com/en/news/securing_ios_apps_patching_binaries/
- Encrypted code modules prevent modifications of the SDK. The link below describes the code module encryption in more detail: https://github.com/project-imas/encrypted_code_modules/

3.5.12 Run-time data protection

This requirement can be fulfilled using obfuscation. For more information on this, see chapter 3.5.9 Protection from reverse engineering.

3.5.13 Sensitive data in memory and temporary storage locations

In SDK scope, this would mean the following:

- *AuthenticationRequestParameters* class is created by SDK but ownership is transferred to 3DSRequestor.
- *ChallengeParameters* class is passed as parameter to SDK from 3DSRequestor and SDK not owned by SDK.

- *Transaction key pair*. Every transaction generates an ephemeral key pair which should be cleared.

Android

String objects should be avoided and use char array which is mutable and can be replaced with random values. However, the Core Specification states the getters and possible setters are defined as String.

Transaction has to generate an ephemeral key pair. java.security.KeyPair doesn't provide a way to destroy the private key from memory.

iOS

Securing memory has 3rd party implementation:
https://github.com/project-imas/memory-security

3.5.14 Encryption of stored data

The SDK must not store any transactional data locally, including the transient keys.

3.5.15 Generation of pseudo random numbers

Identifiers defined in the Core Specification are in form of UUID. Since each platform can directly produce good quality random UUIDs, they can be used directly. On the other hand, RFC4122 specifically says the following:

“Do not assume that UUIDs are hard to guess; they should not be used as security capabilities (identifiers whose mere possession grants access), for example. A predictable random number source will exacerbate the situation.”

The identifiers, as they are defined as 4122 UUIDs can’t be relied on being securely random.

Android

As described by https://developer.android.com/reference/java/util/UUID.html#randomUUID() Android uses cryptographically strong pseudo random number generator.

```
UUID.randomUUID().toString();
```

iOS

The random number generation in iOS can be done with the following code:

```
[[[NSUUID UUID] UUIDString] lowercaseString];
```

Windows 10 Mobile

The random number generation in iOS can be done with the following code:

```
Guid.NewGuid().ToString();
```

Random numbers used in the encryption keys are using the secure random implementations of the platform directly. See the examples in Diffie-Hellman process.

Android

As described in https://developer.android.com/reference/java/security/SecureRandom.html, the java.security.SecureRandom is a cryptographically strong random number which minimally complies with the statistical random number generator tests specified in FIPS 140-2, Security Requirements for Cryptographic Modules.

```
SecureRandom random = new SecureRandom();
```

iOS

The SecRandomCopyBytes interface from the iOS Security Framework (https://developer.apple.com/documentation/security/randomization_services) generates an array of cryptographically secure random bytes. The Security Framework uses the corecrypto library which has been submitted for validation of compliance to FIPS 140-2, as described in https://developer.apple.com/security/.

```
SecRandomCopyBytes(kSecRandomDefault, bytes, randomBytes);
```

4 User Interface implementation

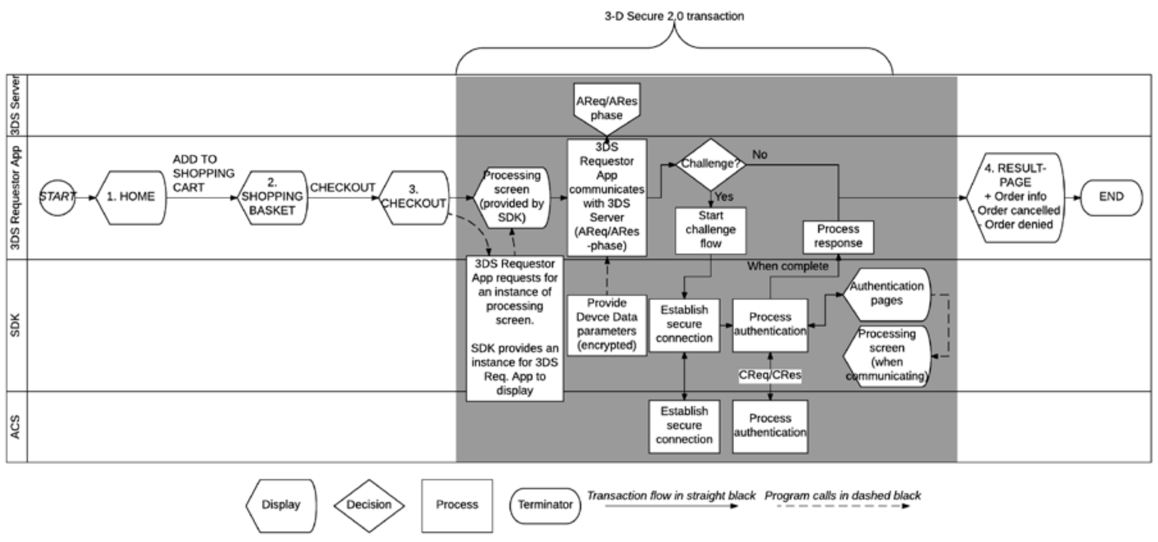
This chapter describes the implementation from a user-interface viewpoint. While the user experience covers the whole flow from the launch of the 3DS Requestor App to a successful purchase transaction, this chapter has parts of the 3DS Requestor App included for clarity.

4.1 Navigation

The following figure demonstrates the navigation flow of the 3DS Requestor App and the SDK. The navigation flow described below is indicative and demonstrates a typical purchase transaction on a mobile app.

The figure below is an overview of the whole purchase transaction. 3-D Secure transaction within the whole transaction is highlighted in gray.

Figure 5 Navigation map of the 3DS Requestor App and SDK in SMS-OTP flow


3-D Secure 2.0 transaction
3DS Server
3DS Requestor App
SDK
ACS
AReq/ARes phase
START
1. HOME
ADD TO SHOPPING CART
2. SHOPPING BASKET
CHECKOUT
3. CHECKOUT
Processing screen (provided by SDK)
3DS Requestor App communicates with 3DS Server (AReq/ARes -phase)
Challenge?
No
Yes
Start challenge flow
Process response
4. RESULT-PAGE + Order info Order cancelled Order denied
END
3DS Requestor App requests for an instance of processing screen.
SDK provides an instance for 3DS Req. App to display
Provide Devce Data parameters (encrypted)
Establish secure connection
Process authentication
When complete
Authentication pages
Processing screen (when communicating)
CReq/CRes
Establish secure connection
Process authentication
Display
Decision
Process
Terminator
Transaction flow in straight black
Program calls in dashed black

From an SDK viewpoint the navigation is rather simple.

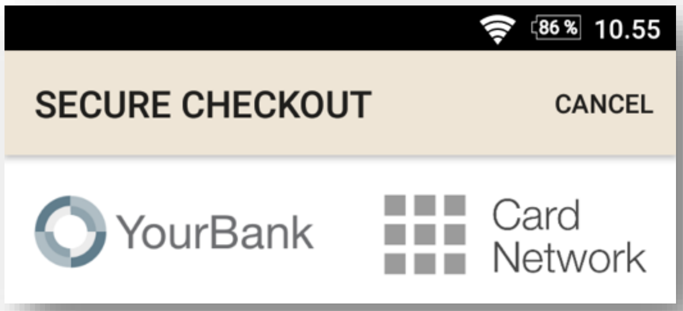
- When the 3-D Secure transaction starts, the 3DS Requestor App fetches an instance of the Progress View from the SDK and displays it to the user. The Progress View is owned and controlled by the SDK, displayed by 3DS Requestor App.
- 3DS Requestor App communicates with 3DS Server, which performs the AReq/ARes-phase. The Progress View is displayed during this phase.
- The transaction moves to the SDK if challenge is to be performed. While SDK communicates with ACS, the Progress View is displayed.
- The user may exit the SDK screens in two ways 1) by clicking on the Back button (on operating level in Android and Windows 10 Mobile) OR 2) by clicking on the Cancel button on upper right corner
- After the challenge has been completed, the transaction returns back to the 3DS Requestor App and the transaction result (confirmation, thank you, etc.) is displayed.

4.1.1 Cancel vs. Back button

One major difference in the platforms Android, iOS, and Windows 10 Mobile user interface is the back button. While Android and Windows 10 Mobile operating systems offer a back button (at the bottom of the screen), the iOS does not. Instead, in iOS applications, the cancel functionality is often implemented on the application level at the upper right corner.

For clarity and consistency among the different platforms, the Cancel button has been implemented on all platforms.

Figure 6 Cancel button on top right corner



4.2 Examples of pages shown by the SDK

This chapter gives examples of pages implemented in the SDK. The screen captures are from the proof-of-concept implementation commissioned by EMVCo and are intended to give overview of possible implementation. The screen captures are not, however, intended to act as specifications or design requirements.

This chapter gives examples of pages implemented in the SDK. The screen captures are made with a standard Android device with a display size of 4.6 inches, 720 x 1280 pixels.

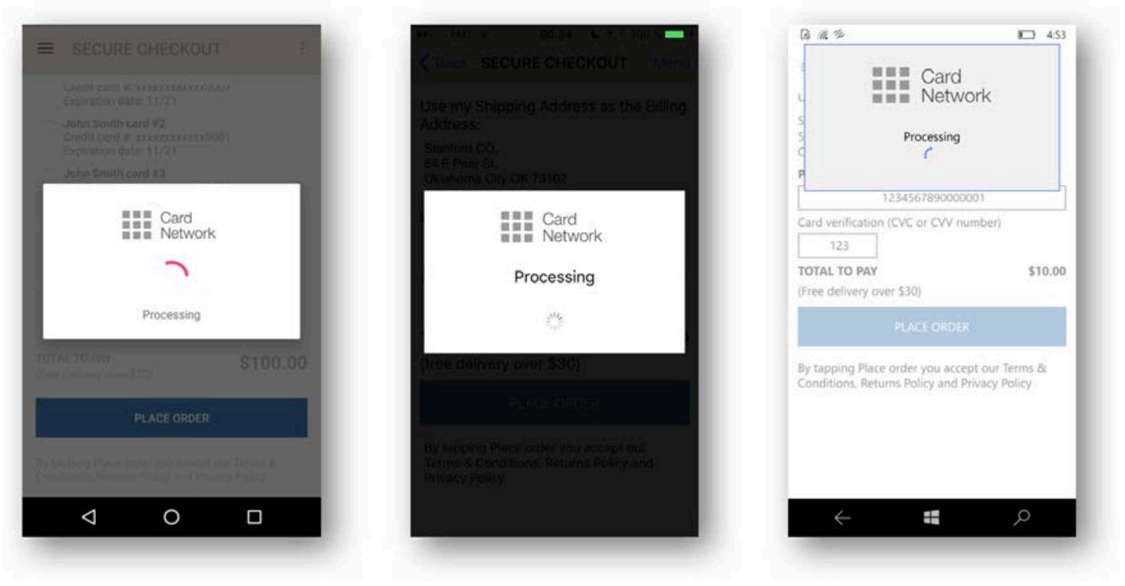
> **Note that the screen sizes may vary and the pages should be tested in various devices with different display sizes. Also, note that the guidelines for the DS and issuer logos may vary.**

4.2.1 Progress View

One of the functions of the SDK is to provide a processing-page while the 3DS Requestor App or the SDK is communicating with other 3-D Secure components.

The Core Specification gives high-level guidelines how the Progress View should be implemented and what graphical components it must and may hold. In this example, it is an overlay with black background above the application.

Figure 7 Progress Views on different platforms



As seen by the example above, the Progress Views are implemented as an overlay on the current application page content. The layout contains the following components.

- Payment System logo
- Spinning symbol commonly used in the platform
- Text "Processing"

Depending on transaction and network, the need for the Progress View may be very short. To avoid flickering effect, the Core Specification states that the Progress View must be visible for at least 2 seconds.

The examples in Figure 7 show the the Progress View examples when the 3DS Requestor Environment communicates with the 3DS Server. During the challenge flow when the SDK communicates with the ACS, only the progress animation (e.g. spinning wheel, as above) is shown as an overlay above the challenge dialogue without the payment system logo.

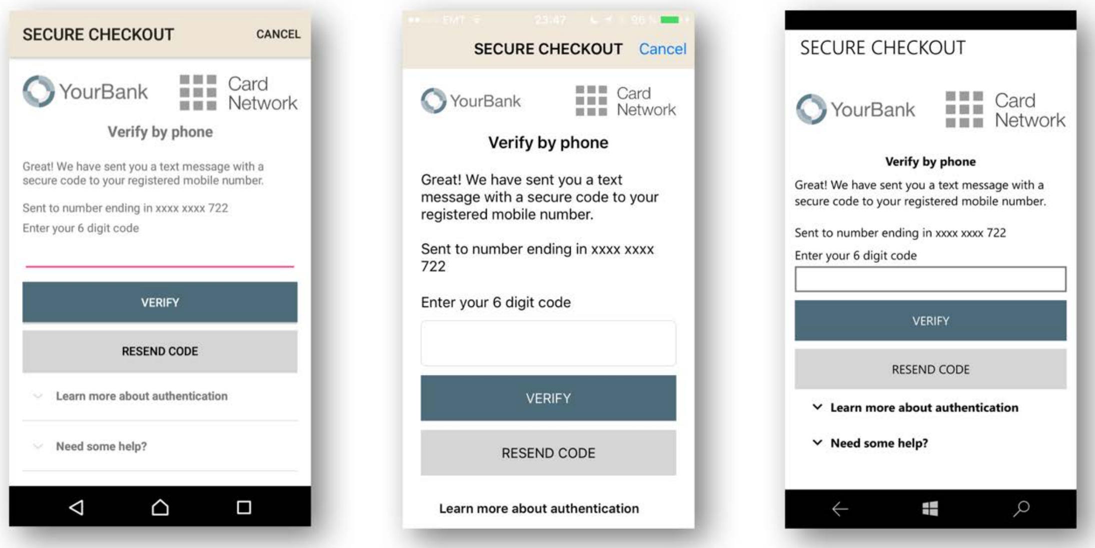
Note that the Payment System logo guidelines will come from each Payment System and are not specified in the Core Specification.

4.2.2 Native OTP Page

Native pages are implemented using the UI-components offered by the platform. Thus, the pages will look slightly different depending on the platform.

The examples below demonstrate how the OTP pages should look like. In these examples, the SMS channel has been used to deliver the One Time Passcode to the user. The page should inform the user how the OTP is delivered / where the user can find the code.

Figure 8 Examples of a native SMS-OTP pages



Verify by phone

Great! We have sent you a text message with a secure code to your registered mobile number.

Sent to number ending in xxxx xxxx 722

Enter your 6 digit code

Content of the challenge pages originates from the ACS

The Payment System and Issuer logos are given to the SDK in CRes-message.

- psImage holds URLs to the Payment System logos of different sizes (medium, high, extraHigh)
- issuerImage holds URLs to the Issuer logos of different sizes (medium, high, extraHigh)

The text on the challenge pages originates from the ACS. ACS can use the following fields in CRes message to transmit the content.

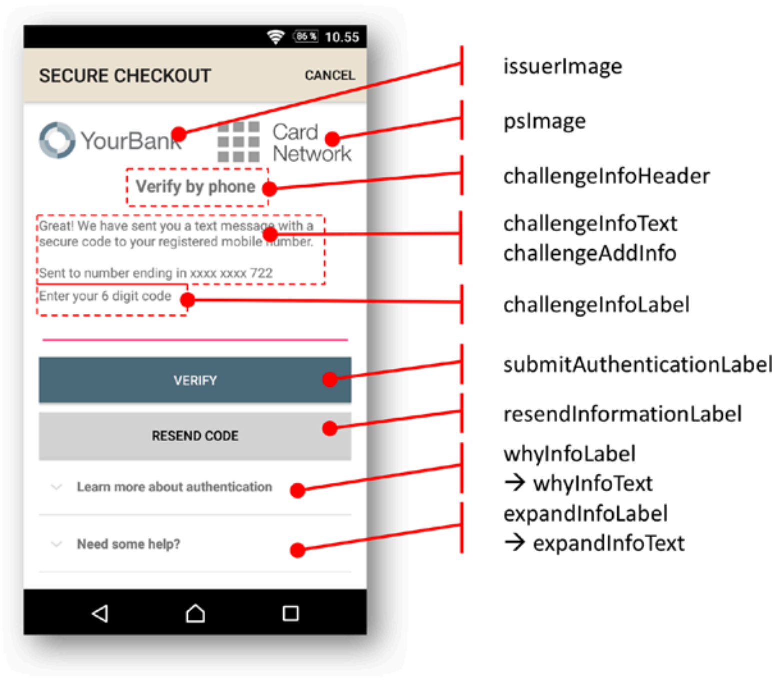
- challengeInfoHeader
- challengeInfoText
- chalengeInfoLabel
- submitAuthenticationLabel
- resendInformationLabel
- challengeInfoTextIndicator (if present, a warning icon or similar visual indicator is shown)

For the expandable info texts on the bottom of the pages, the following fields can be used:

- whyInfoLabel
- whyInfoText
- expandInfoLabel
- expandInfoText

The image below shows the positioning of the text labels on the OTP page.

Figure 9 Usage of CRes-fields on SDK authentication pages



4.2.3 HTML OTP Page

Another way for the ACS to present an OTP page is to deliver the entire content of the page as HTML to the SDK. The SDK displays the page content in a WebView-component. The Core Specification gives clear guidelines on how the HTML is to be formed.

- One HTML file with CSS and images embedded
- Images are embedded in Base64 in the HTML
- The HTML form must have only one target, `HTTPS://EMV3DS/challenge`
- The form may not communicate directly to the ACS; all communication is through the secure CReq/CRes-channel
- The HTML may not contain any external references
- Some versions of Android that use Chromium WebView do not allow navigation to the top frame (ex. <a href="#top"…>), which would prevent some css-based actions from working.

Below is an example of a possible HTML page implementation.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
  <head>
    <meta name="viewport" content="user-scalable=no"/>
    <title>Challenge</title>
    <style>
      [STYLESHEET GOES HERE]
    </style>
  </head>
  <body class="browser">
    <div class="main_div">
    <p>
      <img src="[ISSUER LOGO IN BASE64]" class="logo bank">
```

```
        <img src="[PAYMENT SYSTEM LOGO IN BASE64]" class="logo network">
     </p>

     <h1>Verify by phone</h1>
     <div class="text">
        Great! We have sent you a text message with a secure code
        to your registered phone number.
     </div>
     <div class="text">
        Sent to number ending in
        <span class="masked">[PHONE NUMBER]</span>.
     </div>

     <form class="form_area" action="https://EMV3DS/challenge"
method="GET">
        <div class="prompt">
          Enter your 6 digit code
        </div>
             <input type="number" name="challenge"
                    class="textfield" pattern="[0-9]*">
             <button type="submit" name="submit" value="verify"
  class="button">VERIFY</button>
        <button type="submit" name="submit" value="resend"
                class="button_inverted">Resend code</button>
     </form>

     <hr/>

  </div>
</div>
</body>
</html>
```

Intercepting user input value and action

The SDK intercepts the form action and delivers the content to the ACS in `challengeHTMLDataEntry` field of CReq-message.

The form example above would lead to the following request with challenge entry of `123456`.

`https://emv3ds/challenge?`**`challenge=123456&submit=verify`**

Of the above, `challenge=123456&submit=verify` would be submitted in the `challengeHTMLDataEntry` field.

Below are examples of SDK implementation to intercept the form entry.

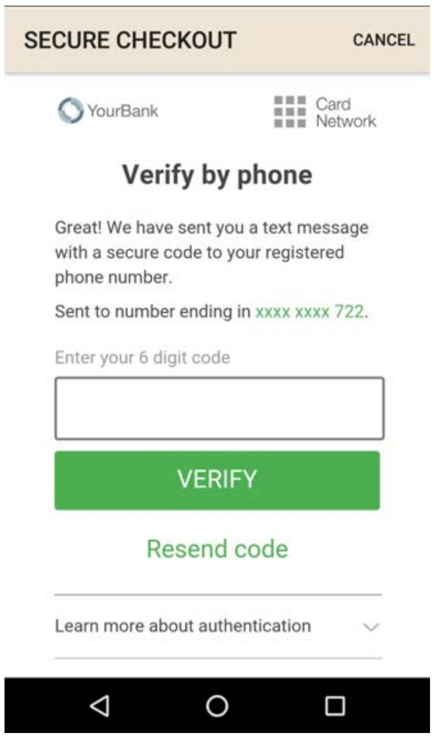
Android

```
public static final String HtmlVerify = "HTTPS://EMV3DS/challenge";

private WebResourceResponse intercept(Uri requestUri) {
  String uri = requestUri.toString();
  boolean intercept =
uri.toLowerCase().startsWith(ProtocolConstants.HtmlVerify.toLowerCase());
  log.debug("WebView shouldInterceptRequest uri: " + intercept);

  if (intercept) {
    String htmlData = uri.startsWith(ProtocolConstants.HtmlVerify) ?
         uri.replace(ProtocolConstants.HtmlVerify + "?", "")
         : uri.replace(ProtocolConstants.HtmlVerify.toLowerCase()
                   "?", "");
    challengeHtmlDataEntered(htmlData);
  }

  if (uri.startsWith("data:text/html")) {
    return null;
  } else {
    return new WebResourceResponse("text/html", "UTF-8", new
ByteArrayInputStream(new byte[0]));
  }
}
```

iOS

```
- (BOOL)webView:(UIWebView *)webView
shouldStartLoadWithRequest:(NSURLRequest *)request
 navigationType:(UIWebViewNavigationType)navigationType {

  NSURL *url = request.URL;
  SDKLogD(@"webView shouldStartLoadWithRequest url");

  NSURLComponents *urlComponents = [NSURLComponents
componentsWithString:[url absoluteString]];
  NSMutableDictionary *params = [[NSMutableDictionary alloc] init];
  for(NSURLQueryItem *item in urlComponents.queryItems)
  {
   [params setValue:item.value forKey:item.name];
  }

  NSString *urlString = [url absoluteString];
  if ([[urlString lowercaseString] hasPrefix:[KHtmlVerify
lowercaseString]]) {
   SDKLogD(@"3DS20URL detected");
   NSString *htmlData = [urlString hasPrefix:KHtmlVerify] ?
     [urlString stringByReplacingOccurrencesOfString:[KHtmlVerify
stringByAppendingString:@"?"] withString:@""]
     : [urlString stringByReplacingOccurrencesOfString:[[KHtmlVerify
lowercaseString] stringByAppendingString:@"?"] withString:@""];
   [self challengeHtmlDataEntered:htmlData];
  }
  return YES;
}
```

Figure 10 An example of HTML implementation


SECURE CHECKOUT
CANCEL
YourBank
Card Network
Verify by phone
Great! We have sent you a text message with a secure code to your registered phone number.
Sent to number ending in xxxx xxxx 722.
Enter your 6 digit code
VERIFY
Resend code
Learn more about authentication

Size of the HTML

The Core Specification defines the maximum size for the field acsHTML

The example HTML page above with Base64 images (Issuer and Payment System logos) and CSS embedded has a total size of about 40kB. It is broken down into the following:

- Images are about 12-15kB each
- CSS is about 4kB
- HTML is about 4kB

Note that the image sizes may vary depending on the image format. PNG-formatted images giving benefits of transparency may take up to twice the space of JPG-images.

The HTML page may not include JavaScript or other such scripting. The accordion effect on the "*Learn more about authentication*" and "*Need some help?*" UI blocks may be implemented using CSS. However, this approach may not work with some versions of Android that use the Chromium WebKit.

The example below illustrates one way of implementation.

```
<div class="accordion_block" id="#top">
     <a href="#one" class="accordion_header">
     <span  class="accordion_header_text">
        Learn more about authentication </span>
        <span  class="accoridon_icon">
           <img src="data:image/png;base64,[base64 image goeshere]">
           </span>
     </a>
     <div id="one" class="accordion">
        [Info text 1 goes here..]
        <a href="#top" class="close_link">Close</a>
```

```
    </p></div>

    <hr>

    <div class="accordion_block" id="#top">
    <a href="#two" class="accordion_header">
      <span class="accordion_header_text">Need some help?</span>
      <span  class="accoridon_icon">
        <img src="data:image/png;base64,[base64 image goeshere]">
      </span>
    </a>
    <div id="two" class="accordion">
      [Info text 2 goes here..]
      <a href="#top" class="close_link">Close</a>
    </div>
  </div>
</div>
```

The `acsHTMLRefresh` field, if present, can replace the WebView contents when the SDK returns to foreground. This allows the challenge view to be updated for HTML OOB challenge flows to better inform the user of the current state of the transaction.

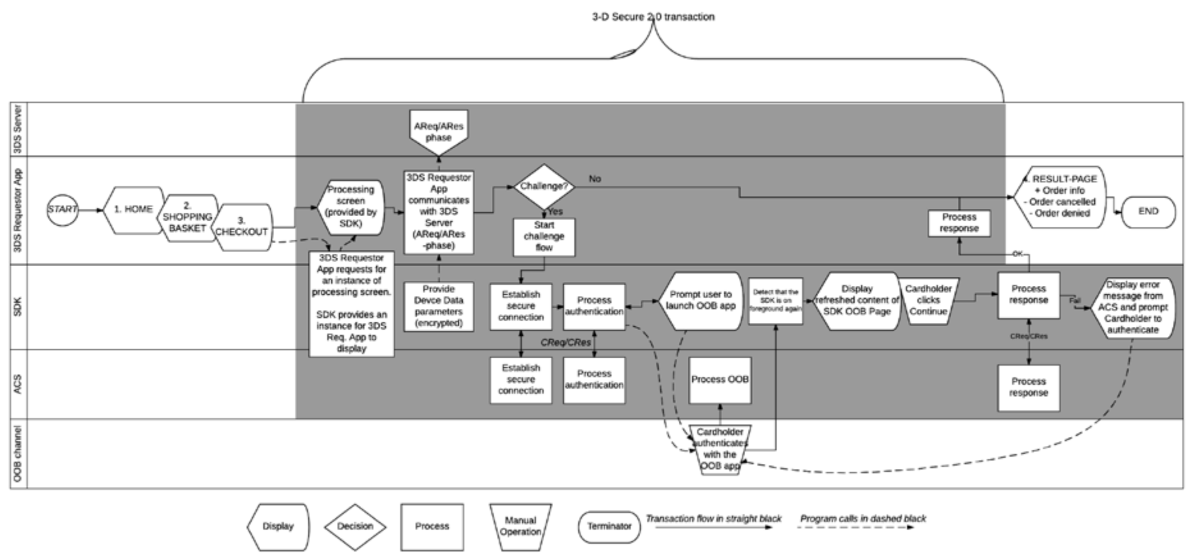
4.2.4 Out of Band

In the Out of Band challenge flow the authentication step happens outside the SDK, e.g. in another application.

The figure below illustrates the transaction flow. In addition to the previous components, the Out of Band authentication app is new in this flow. The specification does not specify the details of the application.

If the transaction should proceed to an OOB authentication, the cardholder is prompted to open the OOB application manually. After returning from the OOB app back to the SDK OOB page, the SDK may refresh the displayed content when it detects that it is taken from background to foreground.

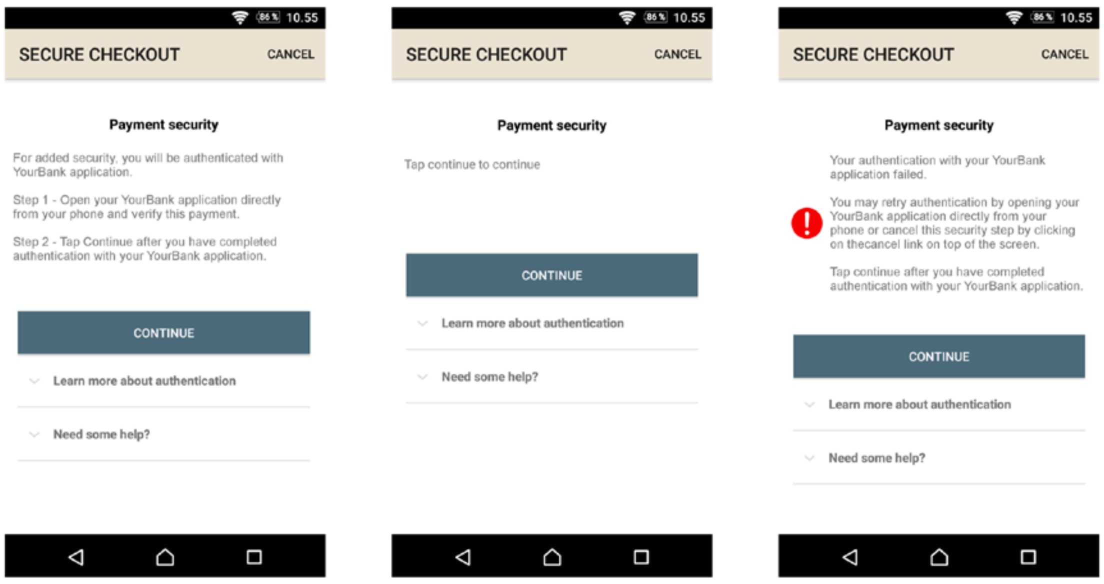
Figure 11 Simplified transaction flow (Out of Band)


3-D Secure 2.0 transaction
3DS Server
3DS Requestor App
SDK
ACS
OOB channel
START
1. HOME
2. SHOPPING BASKET
3. CHECKOUT
Processing screen (provided by SDK)
3DS Requestor App requests for an instance of processing screen. SDK provides an instance for 3DS Req. App to display
AReq/ARes phase
3DS Requestor App communicates with 3DS Server (AReq/ARes -phase)
Provide Devce Data parameters (encrypted)
Challenge?
No
Yes
Start challenge flow
Establish secure connection
Process authentication
CReq/CRes
Prompt user to launch OOB app
Detect that the SDK is on foreground again
Display refreshed content of SDK OOB Page
Cardholder clicks Continue
Process OOB
Cardholder authenticates with the OOB app
Process response
OK
Fail
Display error message from ACS and prompt Cardholder to authenticate
RESULT-PAGE + Order info - Order cancelled - Order denied
END
Display
Decision
Process
Manual Operation
Terminator
Transaction flow in straight black
Program calls in dashed black

The examples below are from the proof-of-concept project where different flows were experimented. Figure 12 shows three different pages in Native implementation. The App-based HTML flow wirks similarly but with relevant HTML data elements as specified for CRes-messages.

- The leftmost page is shown to the cardholder first to inform about the upcoming OOB challenge
- After the OOB challenge has taken place and the cardholder has closed the OOB application / moved back to 3DS Requestor App, the page on the middle is shown. The content of the page has been transmitted in `challengeAddInfo` in the first CRes-message. See below for code samples to detect when the SDK page comes to foreground.
- After the cardholder clicks on Continue-button, the SDK performs a CReq/CRes-dialogue with ACS. If the OOB challenge has not been completed successfully, the ACS returns page content for the rightmost page and the cardholder may try authentication again.

Figure 12 Example of SDK page asking to open the OOB application



86% 10.55

SECURE CHECKOUT CANCEL

Payment security

For added security, you will be authenticated with YourBank application.

Step 1 - Open your YourBank application directly from your phone and verify this payment.

Step 2 - Tap Continue after you have completed authentication with your YourBank application.

CONTINUE

Learn more about authentication

Need some help?

86% 10.55

SECURE CHECKOUT CANCEL

Payment security

Tap continue to continue

CONTINUE

Learn more about authentication

Need some help?

86% 10.55

SECURE CHECKOUT CANCEL

Payment security

Your authentication with your YourBank application failed.

You may retry authentication by opening your YourBank application directly from your phone or cancel this security step by clicking on thecancel link on top of the screen.

Tap continue after you have completed authentication with your YourBank application.

CONTINUE

Learn more about authentication

Need some help?

Updating the SDK page – Native

After the user returns from the OOB app to the 3DS Requestor app, the SDK OOB Challenge Page is updated to display the `challengeAddInfo` when the SDK detects that it is moved to the foreground.

Updating the SDK page – HTML

After the user returns from the OOB app to the 3DS Requestor app, the HTML WebView content is updated to display the `acsHTMLRefresh` when the SDK detects that it is moved to the foreground.

Code samples

Below are examples of SDK implementation to detect the SDK OOB Challenge Page returning to foreground.

Android

```
    private static final String STATE_REFRESH_UI = "refresh_ui";
    private boolean refreshUI = false;

     @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (savedInstanceState != null) {
            refreshUI = savedInstanceState.getBoolean(STATE_REFRESH_UI,
false);
        }

     :
   }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(STATE_REFRESH_UI, true);
    }

    @Override
    protected void onResume() {
        super.onResume();

        if (refreshUI) {
            //refreshUI if not the first time the activity is created
            // and there is no ongoing request
            log.debug("Refresh UI of OOB challenge");
            refreshUI();
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        refreshUI = true;
    }
```

iOS

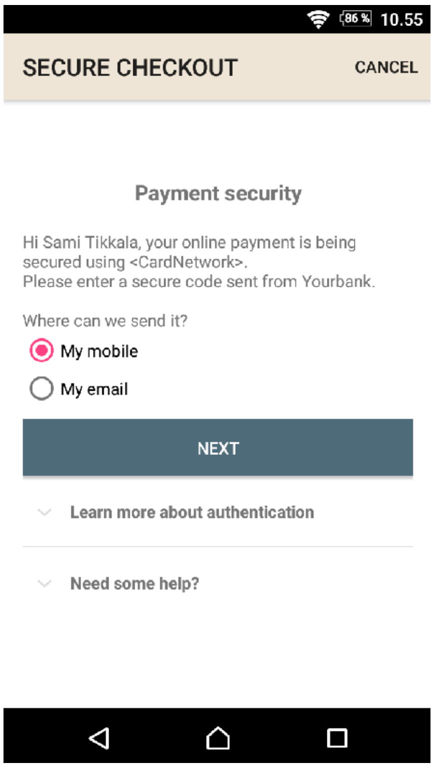
```
- (void)viewDidLoad {
  [super viewDidLoad];
  :
  [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(oobRefreshUI)
name:UIApplicationWillEnterForegroundNotification object:nil];
  [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(oobGoToBackground)
name:UIApplicationDidEnterBackgroundNotification object:nil];
}

- (void)oobGoToBackground {
 [self hideAllViews];
}

- (void)oobRefreshUI {
 SDKLogD(@"OOB refresh UI");
  [self showAllViews];
  :
  [self.view setNeedsLayout];
}
```

4.2.5 Single Select Page

The single select page may be used to give the cardholder a choice during the challenge flow. For example, the user may be asked if the one-time passcode is to be delivered, email or phone. If the ACS holds multiple phone numbers, the user may be asked where the OTP is to be delivered.

The example below illustrates one implementation of the single select page.

Figure 13 Example of a Single Select page



Payment security

Hi Sami Tikkala, your online payment is being secured using <CardNetwork>.
Please enter a secure code sent from Yourbank.

Where can we send it?

[✓ My mobile]

My email

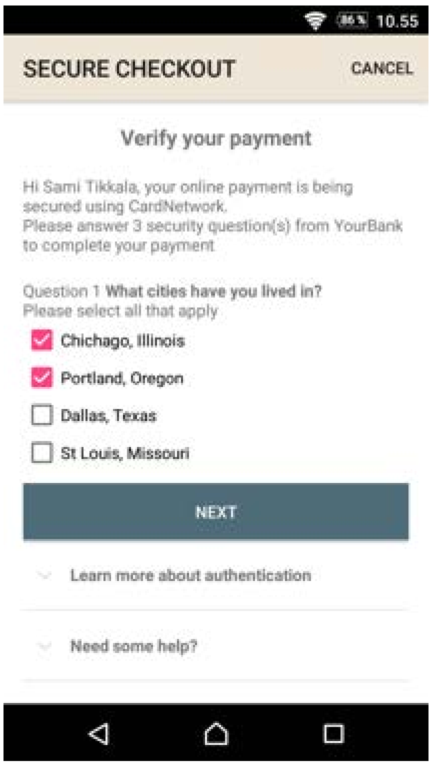
The content of the page is delivered to the SDK in CRes-message by the ACS. The response is sent back in CReq-message.

4.2.6 Multi-select

The Multi-select page may be used to perform multiple choice questions, even knowledge-based authentication.

The example below is a flow where answering correctly to the challenge would lead to a successful authentication transaction, failing to answer the challenge would lead to a stronger authentication method.

Figure 14 Example of a Multi-select page



Verify your payment

Hi Sami Tikkala, your online payment is being secured using CardNetwork.
Please answer 3 security question(s) from YourBank to complete your payment

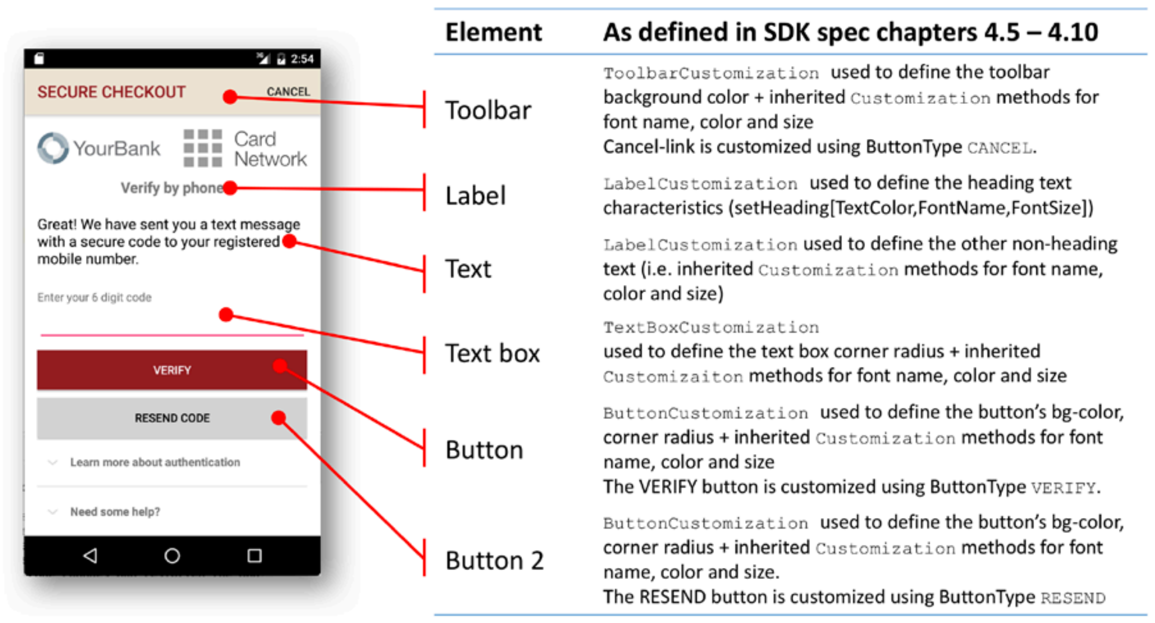
Question 1 **What cities have you lived in?**
Please select all that apply

- [x] Chichago, Illinois
- [x] Portland, Oregon
- [ ] Dallas, Texas
- [ ] St Louis, Missouri

4.3 UI Customization

The SDK specification defines the functions that can be used by the 3DS Requestor to customize the user interface of the authentication screens. The customization is somewhat limited and can be applied only to the specific parts of the screen.
The figure below describes the UI customization in practice.

Figure 15 Illustration of elements that can be customized on OTP page



Great! We have sent you a text message with a secure code to your registered mobile number.

Element	As defined in SDK spec chapters 4.5 – 4.10
Toolbar	`ToolbarCustomization` used to define the toolbar background color + inherited `Customization` methods for font name, color and size Cancel-link is customized using ButtonType `CANCEL`.
Label	`LabelCustomization` used to define the heading text characteristics (setHeading[TextColor,FontName,FontSize])
Text	`LabelCustomization` used to define the other non-heading text (i.e. inherited `Customization` methods for font name, color and size)
Text box	`TextBoxCustomization` used to define the text box corner radius + inherited `Customizaiton` methods for font name, color and size
Button	`ButtonCustomization` used to define the button's bg-color, corner radius + inherited `Customization` methods for font name, color and size The VERIFY button is customized using ButtonType `VERIFY`.
Button 2	`ButtonCustomization` used to define the button's bg-color, corner radius + inherited `Customization` methods for font name, color and size. The RESEND button is customized using ButtonType `RESEND`

The Specifications do not restrict how the colours may be used. For example, if the font and background colour of a button are set to white, it would appear invisible to the cardholder. When implementing 3DS SDK, it may be a good idea to consider such validations.

4.4 Accessibility

When implementing the user interface, accessibility issues are to be taken into account. The colors and fonts selected for the SDK authentication pages should be selected so that readability and usability will not suffer.

4.4.1 Accessibility and UI Customization

As UI customization allows the 3DS Requestor to modify the user interface parameters of different elements as defined in chapter 4.2. The specification does not define limitations to the user interface regarding colors, font sizes and fonts being used in the authentication screens. However, SDK implementations may restrict the customization to prevent usability and accessibility issues. For example:

- Too small font sizes
- Font and background color on toolbars or buttons are too close making text invisible or hard to read

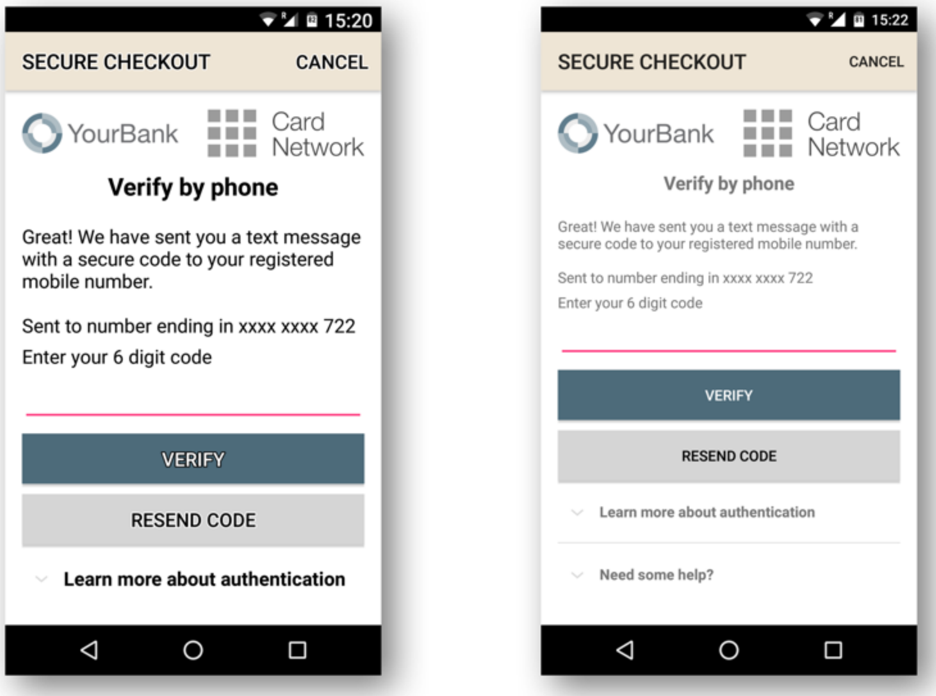
4.4.2 Accessibility settings on mobile devices

Users of Android, iOS and Windows 10 Mobile operating systems have the option to switch on accessibility settings to enhance visibility, for example increasing font sizes and contrast or changing colors.

From development point of view there is not much to do besides testing and verifying that the SDK performs well on operating level accessibility settings.

The figure below shows the OTP authentication page with accessibility settings (larger size and higher contrast) switched on.

Figure 16 Larger font and higher contrast settings on (left) and off (right)



15:20

SECURE CHECKOUT CANCEL

YourBank Card Network

Verify by phone

Great! We have sent you a text message with a secure code to your registered mobile number.

Sent to number ending in xxxx xxxx 722

Enter your 6 digit code

VERIFY

RESEND CODE

Learn more about authentication

15:22

SECURE CHECKOUT CANCEL

YourBank Card Network

Verify by phone

Great! We have sent you a text message with a secure code to your registered mobile number.

Sent to number ending in xxxx xxxx 722

Enter your 6 digit code

VERIFY

RESEND CODE

Learn more about authentication

Need some help?

5 Merchant Implementation Considerations

This chapter describes the use of the SDK implementation from Merchant's viewpoint. The purpose of this chapter is to give a viewpoint to what is required from the 3DS Requestor Application to utilize the SDK.

This chapter has been written with the following assumptions:

- The examples are based on an Android implementation and are indicative, not intended to work directly in real software
- The EMVC0 3DS2.0 SDK is released as an Android archive (`sdk.aar`) and included in the 3DS Requestor Application

5.1 Including the 3-D Secure SDK in 3DS Requestor App Implementation

In order to use the SDK functionality in a 3DS Requestor Application, it must be included in the development environment.

The SDK package is copied into the `app/libs` directory of the Android studio project for the 3DS Requestor Application and the following lines included in the `build.gradle` (Module:app) configuration file:

```
repositories {
  mavenCentral()
  flatDir {
dirs 'libs' //this way we can find the sdk.aar file in libs folder
  }
}
dependencies {
  :
  :
  compile 'com.emvco.mi_sdk:sdk:0.1@aar' //should be the appropriate name
                                            //and version
}
```

As chapter 3.5.9 Protection from reverse engineering suggests, the development environment should have obfuscation (such as ProGuard) tools in place. Depending on the selected tool, additional configuration for these tools may have to be added in the configurations above.

5.2 SDK Initialization

To start using the SDK, the 3DS Requestor App must first initialize it. With initialization the 3DS Requestor App provides the SDK parameters about configuration, UI customization and locale.

More detailed information about the initialization functionality, see the SDK Specification chapter 4.1.1.

3DS Requestor App creates new instances of ThreeDS2Service, ConfigParameters and UICustomization as follows:

```
ThreeDS2Service service = new ThreeDSecurev2Service();
ConfigParameters configParam = new ConfigParameters();
UiCustomization uiConfig = new UiCustomization();
```

5.2.1 Applying UICustomization

Should the 3DS Requestor wish to have its own look-and-feel in the challenge pages, the UICustomization class is used. For example, the following sample would set the challenge page UI as:

- 10px rounded corners on text input filed
- Monospace 18px font as label font
- Sets the toolbar background as dark red with white 28px font
- Sets the Verify button to have 10px rounding on corners and dark red background with white text

```
UiCustomization uiConfig = new UiCustomization();
TextBoxCustomization textBoxCustomization = new TextBoxCustomization();
textBoxCustomization.setCornerRadius(10);
uiConfig.setTextBoxCustomization(textBoxCustomization);

LabelCustomization labelCustomization = new LabelCustomization();
labelCustomization.setTextFontName(Typeface.MONOSPACE.toString());
labelCustomization.setTextFontSize(18);
uiConfig.setLabelCustomization(labelCustomization);

ToolbarCustomization toolbarCustomization = new ToolbarCustomization();
toolbarCustomization.setBackgroundColor("#951728");
toolbarCustomization.setTextColor("#FFFFFF");
toolbarCustomization.setTextFontSize(28);
uiConfig.setToolbarCustomization(toolbarCustomization);

ButtonCustomization buttonCustomization = new ButtonCustomization();
buttonCustomization.setBackgroundColor("#951728");
buttonCustomization.setCornerRadius(10);
buttonCustomization.setTextColor("#FFFFFF");

uiConfig.setButtonCustomization(buttonCustomization,
UiCustomization.ButtonType.VERIFY);
```

Details about the usage of UICustomization can be found in chapter 4.5 in SDK Specification.

5.2.2 Call to initialize ThreeDS2Service

Finally, the 3DS Requestor Application can call the initialize()-method of the ThreeDS2Service object.

```
service.initialize(currentActivity, configParam, locale, uiConfig);
```

Depending on implementation, calling the `initialize()`-method may take some time and cause a short delay. Thus, it is recommended to call it in a separate thread.

After the `initialize()`-method has been called, the 3DS Requestor App may call the `getWarnings()`-method of theThreeDS2Service to query possible warnings generated during initialization. More information on possible warnings can be found in chapter 4.1.5 of the SDK Specification.

5.3 Transaction initiation

The transaction is created using the `createTransaction()`-method of the `ThreeDS2Service` object `service`. This returns an instance of `Transaction`, through which 3DS Requestor App can access the transaction data.

```
Transaction transaction = service.createTransaction(paymentSystemId,
messageVersion);
```

5.3.1 Transaction Message Version

The `createTransaction()`-method of the ThreeDS2Service interface (SDK Spec Section 1.2) has a `messageVersion` parameter that allows the 3DS Requestor App to specify the Secure Protocol Version to be used by the 3DS SDK during the challenge. This means that the CReq `messageVersion` should follow this parameter, and any received CRes should have its `messageVersion` validated against the parameter value.

If the `messageVersion` parameter is null or empty, then the highest protocol version supported by the SDK will be used. If the `messageVersion` parameter passed is not supported by the SDK, then an `InvalidInputException` should be thrown by the SDK.

See SDK Specification Section 8.3 for the protocol versioning requirements.

5.3.2 Progress dialog

While the 3DS Requestor App is communicating with the 3DS Requestor Environment, a progress dialog is displayed. `Transaction` implements a `ProgressDialog` class, which can be accessed through `getProgressView()`-method.

The example below creates `progressDialog` object and displays it in the current activity.

```
ProgressDialog progressDialog =
 transaction.getProgressView(currentActivity);
currentActivity.setSDKProgressDialog(progressDialog);
currentActivity.showSDKProgressView();
```

5.4 Implementing the AReq/ARes-phase

The actual AReq construction is outside the SDK scope and depends on the 3DS Server implementation. The interface between the 3DS Requestor App and 3DS Server (or other access point in 3DS Environment) is not defined in the Core or SDK Specifications.

The SDK offers an `AuthenticationRequestParameters`-class that is used to access needed transaction data.

- Device Data
- SDK Transaction ID

- SDK AppID
- SDK Reference Number
- SDK Ephemeral Public Key
- Message Version

In addition to the above, the 3DS Server will need further transaction data, such as the card number and purchase amount.

The 3DS Requestor App passes the needed information to 3DS Server which performs the AReq/ARes-phase. During this time, the 3DS Requestor App displays the Progress Dialog to the cardholder and waits for a response from 3DS Server.

5.5 Deciding to proceed to Challenge flow

Based on information received from the 3DS Server, the 3DS Requestor App has to decide if a Challenge is needed. If so, the 3DS Requestor App needs to pass the required data to the SDK so that the CReq-message can be constructed.

Request parameters are held in a `ChallengeParameters` object. More information on the ChallengeParameters class can be found in chapter 4.11 of the SDK Specification.

To perform the actual challenge, the 3DS Requestor App calls the `transaction.doChallenge()`-method with the appropriate `ChallengeStatusReceiver`.

The example below shows one way of implementing this phase in the 3DS Requestor App.

```
String transactionStatus =
authenticationResponse.getString(ProtocolConstants.TransactionStatus);

boolean challenge = transactionStatus.contentEquals("C");

ChallengeParameters challengeParameters = new ChallengeParameters();
if (challenge) -->

challengeParameters.set3DSServerTransactionID(authenticationResponse.getStr
ing(ProtocolConstants.ThreeDSServerTransID));

challengeParameters.setAcsTransactionID(authenticationResponse.getString(Pr
otocolConstants.ACSTransID));

challengeParameters.setACSSignedContent(authenticationResponse.getString(Pr
otocolConstants.ACSSignedContent));

challengeParameters.setAcsRenderingType(authenticationResponse.getString(Pr
otocolConstants.ACSRenderingType));

challengeParameters.setAcsRefNumber(authenticationResponse.getString(Protoc
olConstants.ACSReferenceNumber));


transaction.doChallenge(currentActivity, challengeParameters,
     new ChallengeStatusReceiver() {
  @Override
  public void completed(CompletionEvent e) {
   log.debug("<- Completed");
     //At this point, the Merchant app can contact the 3DS Server to
   //determine the result of the challenge
```

```
    }

    @Override
    public void cancelled() {
     log.debug("<- Cancelled");
       //can go to Cancelled view if desired
    }

    @Override
    public void timedout() {
     log.debug("<- timedout");
     currentActivity.hideSDKProgressView();
       //can show error alert
    }

    @Override
    public void protocolError(ProtocolErrorEvent e) {
     log.debug("<- protocolError");
     currentActivity.hideSDKProgressView();
       //can show error alert
    }

    @Override
    public void runtimeError(RuntimeErrorEvent e) {
     log.debug("<- runtimeError");
     currentActivity.hideSDKProgressView();
       //can show error alert
    }
}, timeout
);
```

5.6 Returning to 3DS Requestor App from SDK

After a completed Challenge phase, the 3DS Requestor App can query for the transaction result from the 3DS Server.

Note that the communication between the 3DS Requestor App and 3DS Server is out of scope of the SDK implementation.

Possible values for the transaction result are:

- Y: Order completed successfully / Authentication successful
- A: Order completed successfully / Attempts processing performed
- N: Order denied / Not authenticated (denied)
- R: Order denied / Authentication rejected
- U: Order failed due technical reasons / Authentication could not be performed

***** END OF DOCUMENT *****