Unbound Crypto-of-Things

Developer's Guide for Android

Version 1.4.2007

August 2020

Table of Contents

1. Revision History

The following table shows the changes for each revision of the document.

Version	Date	Description
1.4.2007	August 2020	Added section Encryption Protocol.
1.4	March 2020	Updated the System Requirements.
1.3	January 2019	Updated for version 1.3.
1.2	December 2018	Added section Offline One-Time Password.
1.2	June 2018	Added a note about the *ServerCertificate* parameter in section Digital Signature.
1.2	April 2018	Rebranded.
1.2.1712.19135		Added "Note about API 27 Emulators in Android Studio 3.0"
1.2.1709		Added initParams API - initialization parameters for the SDK
1.2.1708		Refactored SDK modules.See the Install CoT SDK section.
1.2.1706		Separated smart-proxy from the direct-access and simple proxy.New: smart proxy samples and methods to match the new implementation.
1.2.1705		New “Setting URL Parameters” topic. New "LDAP-based Authentication" topic. New “Proxy-Connection Mode” topic. New “CoT Proxy” topic. New: Enhancements to the Sign and Password classes that support Smart CoT Proxy, and sample code that utilizes these capabilities.

2. Installation

2.1. System Requirements

- Android 4.4 and later.
- Android Wear – API 20 and later.

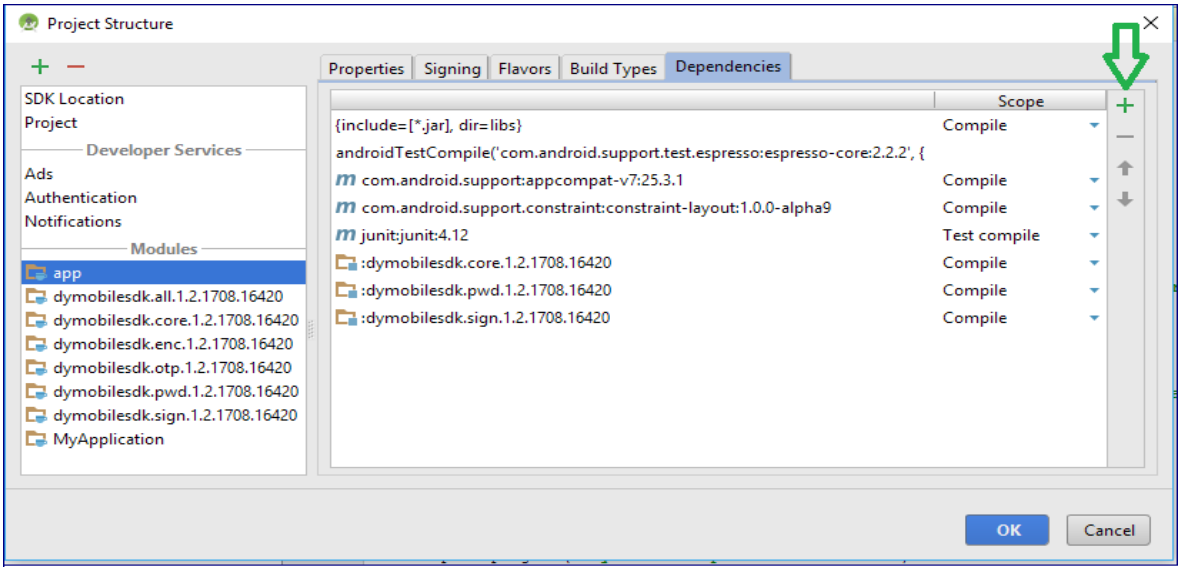
2.2. Install the Unbound CoT SDK

1. Obtain the `Unbound CoT SDK for Android` from Unbound.
2. Extract content from the package, noting the path to the folder where the files are stored. The folder contains the following subfolders:
   - aarA - folder with Unbound Android archive (aar) files.

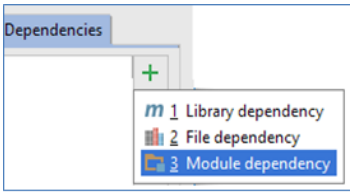
File Name	Description
`dymobilesdk.core.<version>.aar`	The mandatory module.
`dymobilesdk.all.<version>.aar`	Contains the complete suite of Unbound crypto functionality for Android.
`dymobilesdk.sign.<version>.aar`	Enables the use of signing tokens.
`dymobilesdk.enc.<version>.aar`	Enables the use of encryption tokens.
`dymobilesdk.pwd.<version>.aar`	Enables the use of password tokens.
`dymobilesdk.otp.<version>.aar`	Enables the use of OTP tokens.

   - samples - A folder with code samples for the use cases.
   - Javadoc - A folder containing the HTML help. Start from the *index.html* file.

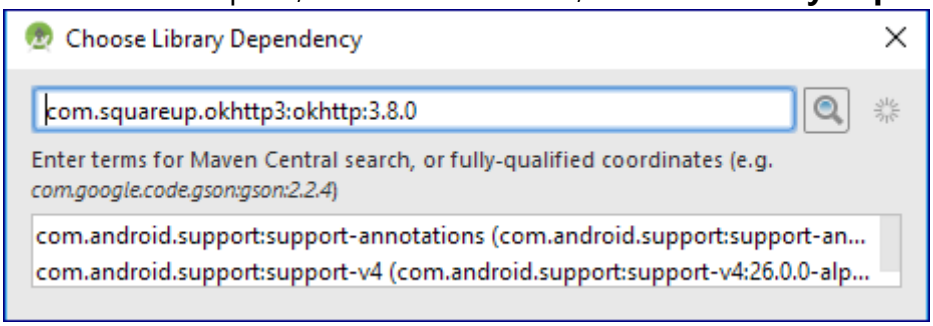
3. Open your project in Android Studio, click **File>Project Structure** and select the **Dependencies**:



4. In the **Modules** pane, choose the **app**, click the "+" button, and click **Module dependency**. The **Choose Modules** pane appears with a list of available modules. Select the required modules and click **OK**.



5. In the **Modules** pane, click the "+" button, and click **Library dependency**.



Use the **Choose Library Dependency** dialog to add the following library:

```
com.squareup.okhttp3:okhttp:3.8.0
```

If you are using the `dymobilesdk.all` or `dymobilesdk.otp` modules, add the following:

```
com.google.android.gms:play-services-wearable:10.0.1
```

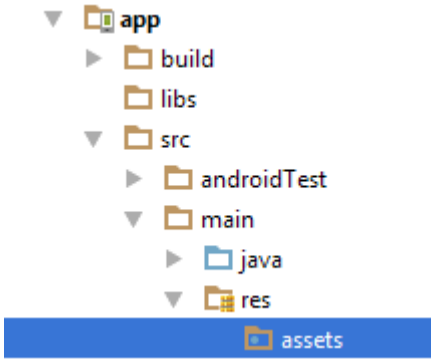
2.3. Add the Certificate

The CoT SDK uses a public key to encrypt the data it is sending to the CoT server. The key and its certificate must be integrated into the application.

The CoT SDK requires a `Base64`-encoded certificate, which is the default encoding of `*.pem` certificates. All other formats must be converted to `Base64`.

Perform the following to integrate the certificate in your IDE.

1. Navigate to **app** > **New** > **Android Resource** and, if needed, add an assets folder:

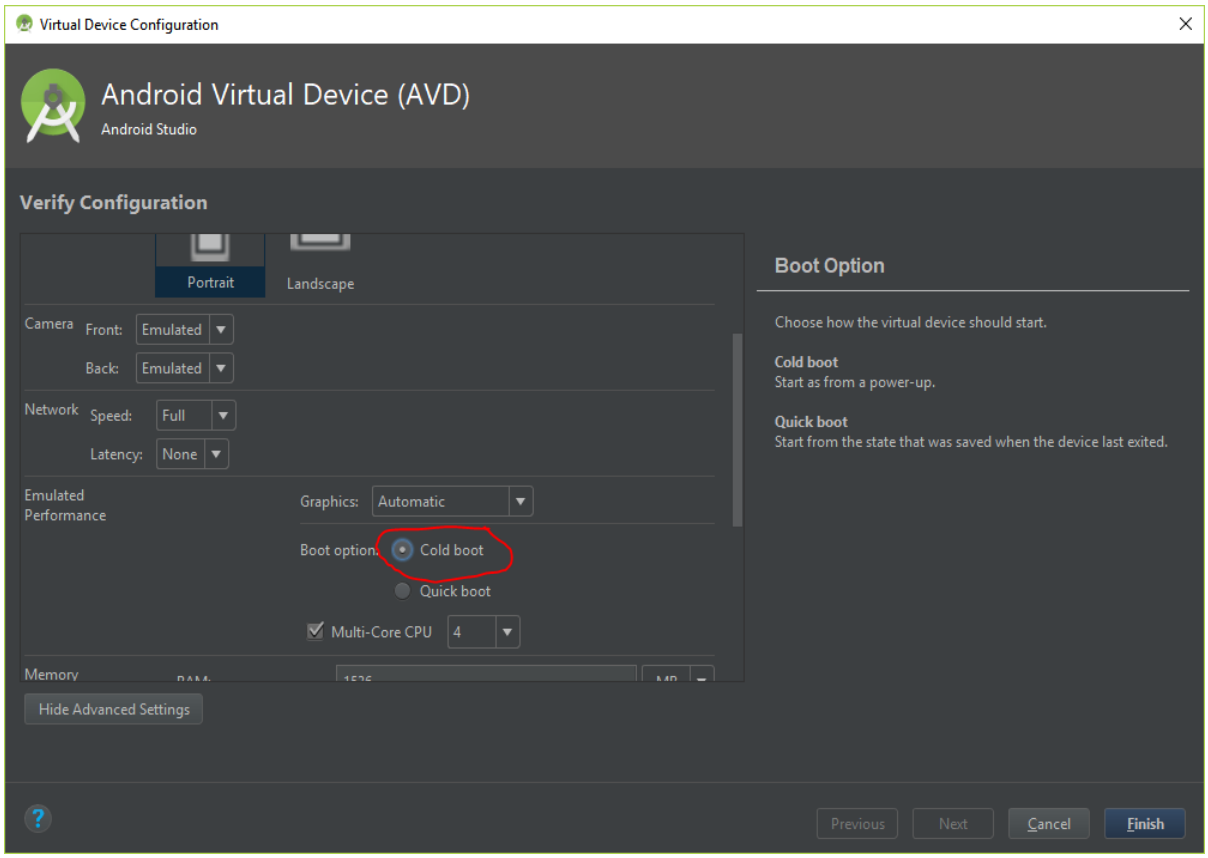


2. Add the Base64-encoded certificate file to the project's assets folder and click **Sync the Project with Gradle Files**.

2.4. Using API 27 Emulators in Android Studio 3.0

There are known issues with the Android Key Store when the emulator is configured to use "Quick boot".

When using an emulator with API 27 or higher, use "Cold boot" as a boot option under "Show Advanced Settings" menu in the emulator settings screen.



3. Introducing Common Concepts

3.1. Direct and Proxy-Based Deployment

A mobile app can address its CoT server in two ways:

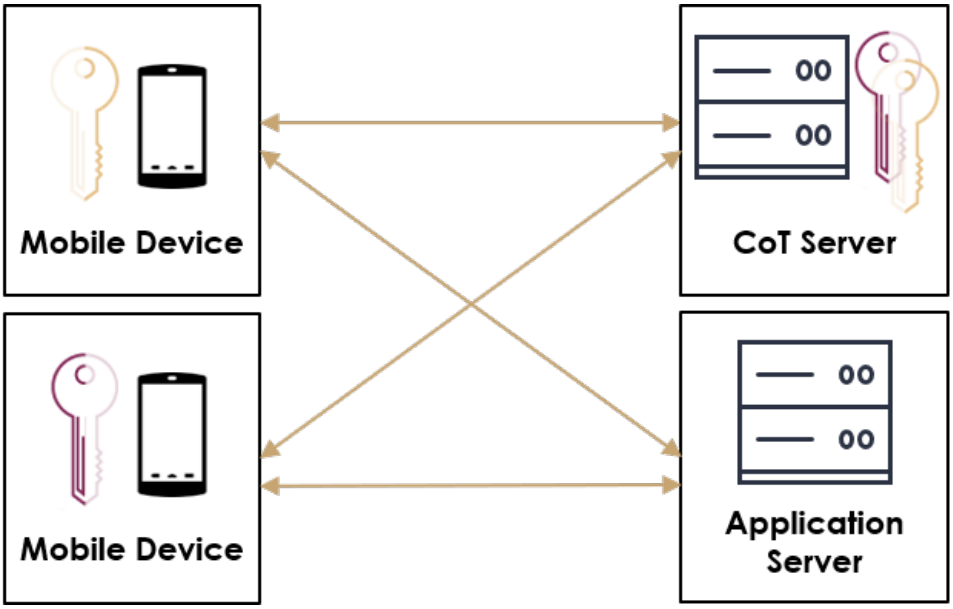
- Direct access - by specifying its URL (or an endpoint on a standard reverse HTTP proxy).
- Via a CoT Proxy - deployed on the app server. The proxy can either be simple or smart.

These three connection methods (**direct**, **simple proxy**, and **smart proxy**) impact the logic of the mobile application:

- Applications that use the direct-access or simple-proxy methods share the same logic and the same methods once they have performed method-specific initialization.
- Applications that use the smart-proxy method requires additional work to perform crypto operations. It uses a different set of methods and a different logic.

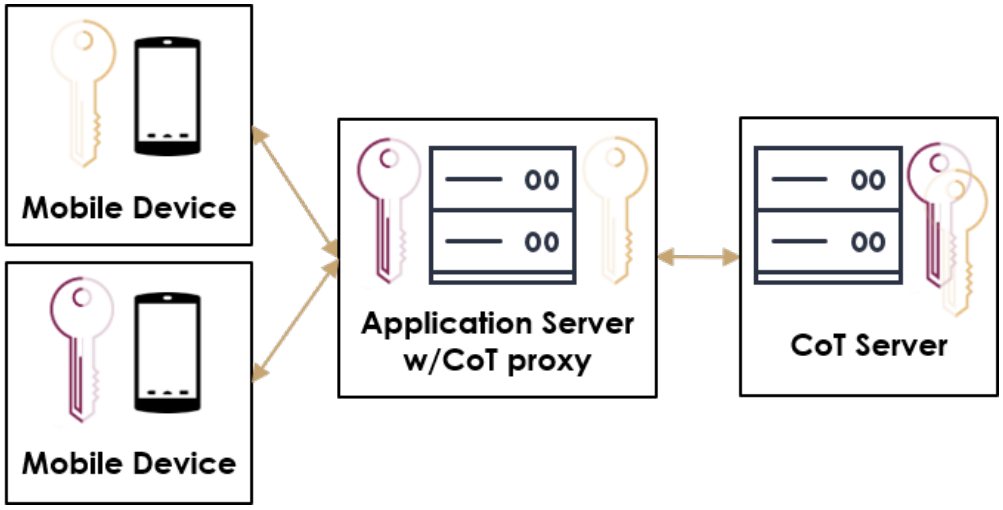
3.1.1. Direct Access

The application has a dedicated URL address to reach the CoT server. The app server is accessed via a different URL address.



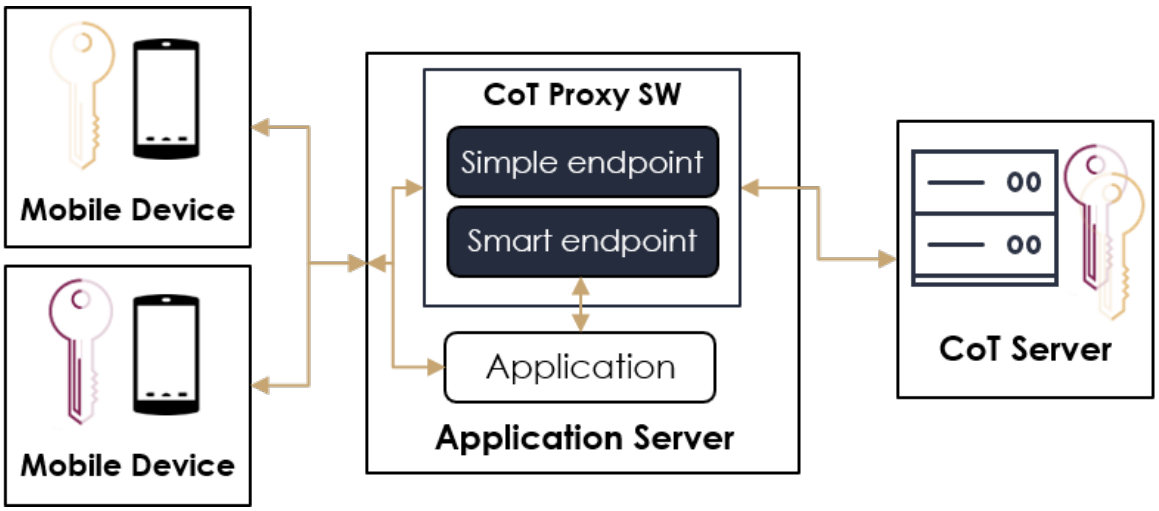
3.1.2. Access Via a Proxy

The mobile application addresses the CoT server via CoT proxy that runs on an application server.



Using CoT Proxy is recommended in configurations where the CoT and the app servers must be reached via the same URL address or when, in addition to simple message forwarding, there is a benefit in involving the app server in multi-party computation performed by the mobile device and the CoT server.

There are two types of CoT Proxy service: Simple and Smart.



Simple proxy – provides a single endpoint on the application server that forwards messages between a mobile device and the CoT server.

Smart proxy – in addition to the simple proxy endpoint, it provides the result of a crypto operation on the application server, as if the operation was performed by the app server. Data can also be encrypted on the client, then start the decryption on the client and get the decrypted value on the smart proxy.

3.2. User Credentials

The CoT SDK supports the following user-authentication methods.

Method	Number of Attempts	Comment
PIN	limited	A salted hash of the PIN is forwarded to the CoT server and compared with the stored one. The server returns an error if they do not match.
Fingerprint	limited	The user must perform a successful fingerprint authentication to use the token. Available only on properly equipped, configured, and initialized devices.
LDAP	enforced by LDAP server	Username and password are verified on the server side by referring the credentials to the configured LDAP server.
Other	unlimited	User authentication is bypassed. This method serves applications that need to authenticate the device itself rather than the device user.

Note
The authentication method is bound to the crypto-token upon its creation. It cannot be changed during the lifetime of the token.

With PIN and fingerprint authentication, the number of attempts is limited. The Unbound crypto service for the specified token is suspended once the allowed number of authentication attempts is exceeded. Following a configured period, this token is re-enabled.

3.2.1. PIN-Based Authentication

The class structure:

```
Java.lang.Object
  com.dyadicsec.mobile.credentials.DYCredentials
    com.dyadicsec.mobile.credentials.DYPinCredentials
```

To configure PIN-based authentication, use the constructor:

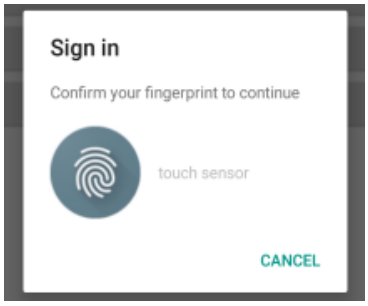
```
String pinCode = input.getText().toString();
new DYPinCredentials(String pinCode)
```

The hash of the PIN is mixed with additional entropy and then forwarded to the CoT server where it compares the provided PIN with the saved one. If the number of the failed PIN attempts exceeds the limit, the server returns an error and stops service requests addressed to this token.

3.2.2. Fingerprint Authentication

The following settings on the device are checked before applying this method:

- The device is running at API level 23 or higher.
- Fingerprint hardware is available on the device.
- A keyguard (`lock screen`) feature is turned on (either pattern or PIN are enabled).
- The user has granted the application permission to access the fingerprint sensor.
- At least one fingerprint has been `enrolled` and is available in the corresponding repository on the device.



Note
The owner of the mobile device `enrolls` her/his fingerprint by scanning the selected finger, as many times as needed, so that the device can capture a consistent image of the finger

For more information, refer to https://developer.android.com/reference/android/hardware/fingerprint/FingerprintManager.html.

The class structure:

```
Java.lang.Object
  com.dyadicsec.mobile.credentials.DYCredentials
    com.dyadicsec.mobile.credentials.DYFingerprintCredentials
```

To configure fingerprint-based authentication, use one of the following constructors:

```
new DYFingerprintCredentials(Activity activity)
new DYFingerprintCredentials(Activity activity, FPDialog fpDialog)
```

Parameters:

- **Activity** - If set to `this` - indicates the current instance of the Android activity. See: https://developer.android.com/reference/android/app/Activity.html
- **FPDialog** - Enables overwriting the default `Unbound Fingerprint Dialog` that guides the user through the process of fingerprint authentication.

  > **Note**
  > The dialog must dismiss itself. The DYMobile SDK does not call the `dismiss` procedure.

  For more information, refer to https://developer.android.com/reference/android/app/DialogFragment.html.

3.2.3. LDAP Authentication

The class structure:

```
java.lang.Object
  com.dyadicsec.mobile.credentials.DYCredentials
    com.dyadicsec.mobile.credentials.DYLDAPCredentials
```

To configure LDAP-based authentication, use the constructor:

```
new DYLDAPCredentials(java.lang.String username, java.lang.String password)
```

For example:

```
LDAP_USENAME = "ad_domain_name\\username";
LDAP_PASSWORD = "password";
new DYLDAPCredentials(LDAP_USENAME, LDAP_PASSWORD)
```

The required format of the username in the provided credentials (username, password) depends on the configuration of the LDAP service access on the CoT server. Consult with the admin of the CoT server about which of the following formats should be used in your application.

Format of Username in LDAP credentials

Format of username	Comment
<username>	Just the sAMAccountName
<domain name>\<username>	Also called `NetbiosName\sAMAcoountName`
<username>@<domain-name>.com	Also called `UPN`

Format of username	Comment
(cn=some,cn=ou,dc=domain,dc=com)	Also called `Distinguished Name`

3.2.4. Void Authentication

The `NoCredentials` method is recommended in the following cases:

- What is being authenticated is the device (and not the user).
- A user may proceed without authentication.

The class structure:

```
Java.lang.Object
  com.dyadicsec.mobile.credentials.DYCredentials
    com.dyadicsec.mobile.credentials.DYNoCredentials
```

To bypass user authentication, use the constructor:

```
new DYNoCredentials()
```

4. Using Direct Access and Simple-Proxy

This chapter specifies the sample code, and methods used in its implementation that are provided in the high-level wrapper classes.

> **Note**
> The direct connection and the simple proxy methods share all methods, except the initialization.

4.1. Initialization

4.1.1. Initialization in the Direct Connection Mode

Perform the following steps:

1. Get a reference to the singleton of the CoT SDK using `DYMobile.getInstance()`.
2. Initialize it using `DYMobile.Init` while specifying the URL of the CoT server.

```
public DYStatus init(
          Context context,
          String url,
          Map<String, String> urlParams,
          String domain
          String certificateResourceName,
          Map<String, Object> initParams)
```

Parameters of Init:

- Context - Android application context
- url - URL of the CoT server
- urlParams - Two options:
  - `null` - indicates no parameters
  - A map of key-value pairs in the format of `key1=val1%key2=val2,` etc.
- domain - Name of the token domain on CoT server assigned to handle the application. If `null`, the `default` token domain is used.
- certificateResourceName - The name of the resource that contains an CoT server certificate received from CA.
- initParams - Initialization parameters for the SDK.

4.1.1.1. urlParams – An Option to Carry Additional Data

> **Note**
> The CoT solution does not require forwarding of HTTP parameters. This capability is provided for use by the application layer.

The `name-value` list of parameters can be passed to the CoT server using one of the following methods:

- A list of any name-value pairs appended to the URL.
- A list of any name-value pairs within the HTTP header.
- Credentials only (client-id, client-password) – these are forwarded in the `Authorization` field of the HTTP header in compliance with the `Base Access Authentication` specification (see https://en.wikipedia.org/wiki/Basic_access_authentication).

> **Note**
> These methods cannot be combined.

The required method is identified by the presence of the following `name-value` pair in the list of the `urlParams`:

Method to Pass Parameters	Value of *USE_HEADERS* in the *urlParams* List
A list of any name-value pairs appended to the URL.	"`USE_HEADERS`" not present in the list
A list of any name-value pairs within the HTTP header.	`USE_HEADERS` = `PLAIN`
Credentials (only) in the `Authorization` field of the HTTP header.	`USE_HEADERS` = `BASIC_AUTHENTICATION`

4.1.1.2. Example: Parameters in the URL

The required parameters are:

```
client_id=0a49b4a4-f824-46aa-b148-647a8e2e1d8a
client_secret=pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM
```

The method:

```
Map<String, String> urlParams = new HashMap<>();
urlParams.put("client_id", "0a49b4a4-f824-46aa-b148-647a8e2e1d8a");
urlParams.put("client_secret", "pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM");
```

```
DYStatus status = DYMobile.getInstance().init(
                                getContext(),
                                SERVER_URL,
                                urlParams,
                                DOMAIN_NAME,
                                CERTIFICATE_FILE_NAME);
```

The URL is:

```
<CoT-Server-URL>?\
client_id=0a49b4a4-f824-46aa-b148-647a8e2e1d8a&\
client_secret=pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM
```

4.1.1.3. Example: Parameters in HTTP Header

The method:

```
Map<String, String> urlParams = new HashMap<>();

urlParams.put("client_id", "0a49b4a4-f824-46aa-b148-647a8e2e1d8a");
urlParams.put("client_secret", "pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM");
urlParams.put(DYRestComm.USE_HEADERS, DYRestComm.PLAIN);

DYStatus status = DYMobile.getInstance().init(
                                // truncated – see above
);
```

The resulting HTTP Header:

```
// Content-Type: application/json; charset=utf-8
// Content-Length: XXX
// Host: XXX
// Connection: Keep-Alive
// Accept-Encoding: gzip
// client_id: 0a49b4a4-f824-46aa-b148-647a8e2e1d8a
// client_secret: pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM7tN5fA8oO
```

4.1.1.4. Base-Access Credentials in the HTTP Header

The method:

```
Map<String, String> urlParams = new HashMap<>();

urlParams.put("client_id", "0a49b4a4-f824-46aa-b148-647a8e2e1d8a");
urlParams.put("client_secret", "pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM");
urlParams.put(
```

```
  DYRestComm.USE_HEADERS, DYRestComm.BASIC_AUTHENTICATION);

DYStatus status = DYMobile.getInstance().init(
                              // truncated – see above
);
```

The resulting HTTP header:

```
// Content-Type: application/json; charset=utf-8
// Content-Length: XXX
// Host: XXX
// Connection: Keep-Alive
// Accept-Encoding: gzip
// Authorization: Basic dm9yZGVsOnZvcmRlbA==
```

The names `client_id` and `client_secret` are not present in the header. Instead, their values are combined together, encoded using `Base64` and stored in the `Authorization` field.

4.1.1.5. initParams – Add initialization parameters for the SDK

It is possible to set the user authentication validity period (the time, in seconds, that passed since the user was successfully authenticated with PIN/pattern/password). Once the validity period has passed token operations fail with error DY_EUSER_NOT_ AUTHENTICATED.

To set a custom user authentication validity period:

```
Map<String, Object> initParams = new HashMap<>();
initParams.put("USER_AUTH_DURATION", 60);
DYStatus status = DYMobile.getInstance().init(context, SERVER_URL, urlParams,
domainName , certificateName, initParams);
```

4.1.2. Initialization in the Simple-Proxy Connection Mode

Perform the following steps:

1. Get a reference to the `singleton` of the CoT SDK using `DYMobile.getInstance().`
2. Call `init()` while providing a `ProxyComm` that implements a communication channel with the CoT server via the application server.

The `ProxyComm` is an example of the `DYComm` class implementation.

```
public class MainActivity {
    public final static String CERT = "place_holder";
    public final static String DOMAIN = "place_holder";

    DYMobile.getInstance().init(
           getApplicationContext(),
           new ProxyComm(),
           DOMAIN,
           CERT);
```

Parameters of Init:

- context - Android application context.
- DYComm object - Defines the method of communication with the CoT server.

  > **Note**
  > Unbound provides sample implementation using an object from `ProxyComm` class.

- domain - Name of the token domain on the CoT server assigned to handle the application.
  If set to `null`, the `DEFAULT_DOMAIN` token domain is used.
- certificateResourceName - The name of the resource that contains an CoT server certificate received from CA.

4.1.2.1. ProxyComm Class

`ProxyComm` class is a sample implementation of the `DYComm` class. It defines:

- An URL of the CoT Proxy (for example, the app server URL).
- The single endpoint that provides a standard proxy function.
- The `SendRequest` method.

```
public class ProxyComm extends DYComm {

    /**
     * The address of the proxy.
     * for example http://172.17.0.88:9999/dyadic-mobile-proxy
     */
    private static final String PROXY_SERVER_URL = "place_holder";

    /**
     * the endpoint of the proxy in the proxy server.
     * For example:"/api/proxy"
```

```
     */
    private static final String PROXY_SERVER_ENDPOINT = "place_holder";

    @Override
    public void sendRequest(
       final String operationID,
       final String domain,
       JSONObject jsonRequest,
       final DYResponseListener listener) {
 // deleted
}
```

Parameters of `SendRequest`:

- operationID - ID of operation to be performed by the CoT server.
- domain - Name of token domain assigned to this application.
- jsonRequest - Request in `json` format.
- listener - A call-back procedure that handles the response from the CoT server.

The following is an example of the password enrollment request in the JSON format:

```
{
  "domain": "Domain",
  "operationID": "password\/protect",
  "requestData": "{
    \"encryptedData\":\"AgIAAcGccMQOPenPmJE+",   // truncated
    \"uid\":\"bf7e4d8d-b973-493c-9893-a9bb078f9b5a\",
    \"keyVersion\":1,
    \"signature\":\"MEQCIHl+ ==\"          //truncated
  }"
}
```

4.2. Password Protection

4.2.1. Class Overview

```
java.lang.Object
    com.dyadicsec.mobile.DYSimpleBase
       com.dyadicsec.mobile.tokens.enc.DYPassword
```

The `DYPassword` is a wrapper class for storing and retrieving a single password.

4.2.1.1. protectPassword

Creates a new token that protects the provided password and splits it into two shares.

This method will delete any other existing password tokens.

```
public void protectPassword (
    String username,
    String password,
    String label,
    DYCredentials credentials,
    boolean authenticationToken,
    Map<String, String> parameters,
   DYSignTokenFactory.DYInitTokenListener listener)
```

Important
This method deletes any other tokens of type `Password` on the device — previously defined passwords become invalid.

Parameters:

- username - Name of the user.
- password - The password string provided as-is.
- label - Name that is assigned to the created token.
  Useful for lookup and monitoring in the CoT Admin display.
- credentials - Credentials.
- authenticationToken - `false` – indicates that the password (when retrieved) is provided to the application in the plaintext.
  `true` – the application has been designed to retrieve `encrypted share` from the CoT server and recombine it with the `share` provided by the device.
- parameters - `null` – standard grade encryption or PQC grade encryption (see the note below).
- listener - The callback handler.

To use PQC-grade encryption, pass the following in the `parameters`:

```
Map<String, String> parameters = new HashMap<>();
parameters.put(DYTokenFactory.IS_PQC,"TRUE");
```

4.2.1.2. retrievePassword

```
public void retrievePassword(
     DYCredentials credentials,
     DYPWDToken.DYRetrievePasswordListener listener)
```

Parameters:

- credentials - Credentials.
- listener - The callback handler.

4.2.2. Sample Code

The sample code presents the use of the following three operations: Init, Storage, and Retrieval of the password.

4.2.2.1. Init

This method creates a `singleton` of the CoT SDK using `DYMobile.getInstance()`. See Initialization.

4.2.2.2. Enrollment of the Password

The following sample of code triggers PIN-based authentication and, after verification, stores the provided password while encrypting it.

```
public void onClick(DialogInterface dialog, int whichButton) {
   String pinCode = input.getText().toString();
   DYPassword.getInstance().protectPassword(
     USERNAME,
     txtPassword.getText().toString(),
     "label",
    new DYPinCredentials(pinCode),
    false,// password shall be retrieved as provided
    null,// standard (not PQC) encryption of the password
    new DYPWDTokenFactory.DYInitTokenListener() {
         // deleted
     });
```

4.2.2.3. Retrieval of the Password

The following sample of code triggers PIN-based authentication and, after verification, stores the provided password.

```
public void onClick(DialogInterface dialog, int whichButton) {
    String pinCode = input.getText().toString();

    DYPassword.getInstance().retrievePassword(
       new DYPinCredentials(pinCode),
       new DYPWDToken.DYRetrievePasswordListener() {
     // deleted
    });
}
```

4.3. Digital Signature

4.3.1. Class Overview

```
com.dyadicsec.mobile.tokens.sign
java.lang.Object
     com.dyadicsec.mobile.DYSimpleBase
          com.dyadicsec.mobile.tokens.sign.DYSign
```

The `DYSign` is a wrapper class for working with the Unbound signing tokens. It provides methods for the common signing actions: creation and deletion of the single signing token as well as the signing of the data.

4.3.1.1. createSignToken

Creates the signature token.

> **Note**
> This method deletes all other signature tokens.

```
public void createSignToken(
     String label,
     String username,
     DYCredentials credentials,
     Map< String, String> parameters,
    DYSignTokenFactory.DYInitTokenListener listener)
```

Parameters:

- Label - Label assigned to the token.
- username - Name of the user.
- credentials - Credentials.

- parameters - null – for RSA-based signing.
  For the additional options, see the note below.
- listener - The callback handler.

To use ECDSA-based signing, assign to the `parameters` the following value:

```
Map<String, String> parameters = new HashMap<>();
parameters.put(DYSignTokenFactory.TYPE, SignTokenType.ECDSA.toString());
```

The default crypto-parameters used for signing are:

Parameter	RSA	ECDSA
Padding	PKCS#1	N/A
Hash	SHA256	SHA256
Key Size	2048	P256

4.3.1.2. sign

`Signing` implicitly refers to the signing token created in the previous step.

```
public void sign(
    byte[] dataToSign,
    DYCredentials credentials,
    IDYSignToken.HASH_ALGORITHM hashAlgorithm,
    IDYSignToken.DYSignListener listener)
```

Parameters:

- dataToSign - The byte array to sign.
- credentials - Credentials.
- hashAlgorithm - `null` – the default (SHA256).
  Other values: SHA1, SHA256, SHA384, SHA512.
- listener - The callback handler.

4.3.2. Sample Code

The sample code presents the use of the following three operations:

4.3.2.1. Init

It creates a `singleton` of the CoT SDK using `DYMobile.getInstance()`. See Initialization.

4.3.2.2. Create Signature Token

```
private void CreateSignTokenHandler() {
    txtSignature.setText("");
//  specify type of encryption used by signing
    Map<String, String> params = new HashMap<>();
    params.put(DYSignTokenFactory.TYPE, rdbECDSA.isChecked()?
    IDYSignToken.SignTokenType.ECDSA.toString() :
    IDYSignToken.SignTokenType.RSA.toString());
    DYSign.getInstance().createSignToken(
      "label",
       USERNAME,
       new DYNoCredentials() ,
       params, // RSA or ECDSA encryption
       new DYSignTokenFactory.DYInitTokenListener() {
       @Override
      public void completed(final DYStatus status, final IDYSignToken token) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // deleted
                    }
                }
            });
        }
    });
```

4.3.2.3. Signing

```
private void signHandler() {
   DYSign.getInstance().sign(
        edtTextToSign.getText().toString().getBytes("UTF-8"),
        new DYNoCredentials(),
        null,// Use SHA256
        new IDYSignToken.DYSignListener() {
            @Override
            // deleted
        });
}
```

4.4. Data Encryption

CoT SDK can be used to encrypt data on the device without storing the key used for encryption on the device. Instead, CoT SDK cooperates with the CoT server to create a pair of keys (`shares`). The device keeps one `share` of the pair while the CoT server keeps the other.

The method of encryption is `ECIES` with the `P256` curve and AES-GCM-256.

4.4.1. Class Overview

```
com.dyadicsec.mobile.tokens.enc
java.lang.Object
    com.dyadicsec.mobile.DYSimpleBase
        com.dyadicsec.mobile.tokens.enc.DYEncrypt
```

The `DYEncrypt` is a wrapper class for working with a single Unbound encryption token. It provides methods for the common encryption actions: creation and deletion of the token as well as the encryption and decryption of the data.

4.4.1.1. Create the Encryption Token

```
public void createEncToken(
         String label,
         String username,
         DYCredentials credentials,
         Map<String, String> parameters,
         DYSignTokenFactory.DYInitTokenListener listener)
```

Attention
This method deletes any other tokens of type `Encrypt` on the device

Parameters:

- label - Name assigned to the token.
- username - Name of the user.
- credentials - Credentials.
- parameters - `null` – for `ECIES` with the `P256` curve.
- listener - The callback handler.

4.4.1.2. Encrypt

Encryption implicitly refers to the encryption token created in the previous step.

> **Note**
> Credentials are not required to perform the encryption

```
public byte[] encrypt(byte[] data)
```

Parameters:

- encData - the data to be encrypted

Returns the encrypted data or null if the encryption failed.

4.4.1.3. Decrypt

Encryption implicitly refers to the Encryption Token created in the previous step. Authentication is required.

```
public void decrypt(
        byte[] encData,
      DYCredentials credentials,
      DYEncToken.DYEncTokenListener listener)
```

Decrypt the provided byte array and execute the callback when done.

Parameters:

- encData - the data to be decoded.
- credentials - Credentials – see Create the Encryption Token.
- listener - The callback handler.

> **Note**
> Specification of the credentials must match the one used to create the encryption token.

4.4.2. Sample Code

The sample code presents the use of the following three operations: Init, Encrypt, and Decrypt.

4.4.2.1. Init

It creates a `singleton` of the CoT SDK using `DYMobile.getInstance()`. Refer to Initialization for more details.

4.4.2.2. Encrypt

```
public void encrypt(View v){
    if ((edtClearText.getText().toString().length() != 0) ){
        byte[] clearText = edtClearText.getText().toString().getBytes();
        byte[] cipherText = DYEncrypt.getInstance().encrypt(clearText);

        // deleted
    }
}
```

4.4.2.3. Decrypt

```
public void decrypt(View v){
   String pinCode = input.getText().toString();
    if ((txtCipherText.getText().toString().length() != 0) ){
        byte[] cipherText =
                      DYMobileUtils.hexToBytes(txtCipherText.getText().toString());
        DYEncrypt.getInstance().decrypt(
          cipherText,
          new DYPinCredentials(pinCode),
          new DYEncToken.DYEncTokenListener() {
            @Override
            public void completed(DYStatus dyStatus, final byte[] decryptedBytes) {
                if (dyStatus.getCode() == DYCoreStatus.DY_SUCCEED){
                    runOnUiThread(new Runnable() {
                         // deleted
                        }
                    });
                }
            }
        });
    }else{
        showResults("please encrypt some text before decrypting");
    }
}
```

5. Using Smart-Proxy

> **Disclaimer**
> Communication with the App Server is implemented in the sample code for demonstration and POC purposes only. It does not include handling of errors and must not be used as-is in the production code.

This chapter specifies use cases for the following scenarios:

- Password Protection – uses mix of simple and smart endpoints
- Digital Signature – uses smart entry points only

5.1. Smart Entry Point Processing Paradigm

Each request that uses a smart entry point triggers a three-step session that contains:

1. Create – opens CoT session and prepares data for its use that is divided into two parts:
    - Request – contains data required to perform the operation.
    - Context – contains the CoT SDK data that is carried from step to step. The application is responsible for providing it to the CoT SDK in steps #2 and #3.
2. Update – processes the data received in response from the server in the context created by Step #1.

    > **Note**
    > This step may be performed several times, depending on the functionality provided by the smart endpoint

3. Finalize – performs functions required before closing the session, and closes it and returns the result.

5.2. Bean Classes

Smart proxy methods use the following bean classes to provide the caller with the outcome of the method.

5.2.1. DYProxyRequest

A `DYProxyRequest` bean aggregates the following classes:

Name	Class	Description
serverRequest	JSONObject	JSON request that needs to be forwarded to the CoT server
context	DYContext	An instance of the context that is created in the enrollment step and is carried across all steps needed to implement a crypto operation.
status	DYStatus	Indicates the success or failure to build proxy request

5.2.2. DYUpdateResult

A `DYUpdateResult` bean aggregates the following classes:

Name	Class	Description
additionalRequest	JSONObject	An optional additional JSON request that might need to be sent to the server.
context	DYContext	An instance of the context to be passed between to the update and finalize operations.
status	DYStatus	Indicates the success or failure of the update operation.

5.3. Password Protection

Password Protection provides a sample of mixing smart and simple endpoints in the implementation of a crypto operation:

- The enrollment of the password (`passwordProtect`) does not require to create anything on the application server. Hence, it is implemented as a simple endpoint.
- The retrieval of the password actually occurs on the application server. Hence it is implemented as a smart endpoint.

5.3.1. Password Enroll Method

5.3.1.1. protectPassword

Protect the password with the CoT SDK. It uses the simple endpoint. Hence, it is implemented in one step.

5.3.1.2. Methods of the Password-Retrieve Session

Password must be reassembled from its shares on the application server. Therefore, it uses smart endpoint on the application server. As such, its implementation triggers the 3-step session as described in the introduction of the Using Smart Proxy chapter:

1. **Create**: Asks the SDK to:
    a. Create a context that will be used in the subsequent steps.
    b. Assembly payload of the request to be delivered to the server.
        - Once the SDK assembles the request, the mobile application is responsible for delivering it to the smart endpoint on the application server.
        - Upon reception of the server response, it must proceed to the next step.
2. **Update**: processes the data received in the server response. The processing is carried in the context created in Step #1.
3. **Finalize**: Ask SDK to finalize (close) processing of this context. It returns success/failure status.

Here are the relevant APIs:

1. **createRetrieveRequest**

    It creates:

    - The context used in the subsequent steps.
    - Assemblies request to be delivered to the server.

2. **updateRetrieveRequest**

    Uses the server response to perform the "update" step of the crypto operation within the context set by the 1st step (`createRetrieveRequest`).

3. **finalizeRetrieveRequest**

    Finalizes (completes) password retrieval operation.

    Must be called by the application as the last step in processing password retrieval request.

5.4. Digital Signature

The `DYSign` class provides the following groups of methods:

- Create (enroll) token that is used in the subsequent signatures.
- Use the token to sign the hash of the provided data.
- Refresh the token. This functionality is optional.
- Delete the token.

> **Note**
> The *ServerCertificate* parameter is provided in API initialization function, not in the proxy functions. If you initialize the API without providing the parameter, you get an enrollment message which is not encrypted. Regardless, the default encryption (using TLS) is always available. The encryption provided by the *ServerCertificate* parameter is an extra layer of encryption that is provided on top of standard TLS.

5.4.1. Methods of the Enroll Session

This group of methods implements session that creates token that are used in the subsequent data signing requests.

1. **createEnrollmentRequest**

   Performs Step #1 (create the request).

2. **updateEnrollmentRequest**

   Performs Step #2 (processing the response).

3. **finalizeEnrollment**

   Performs Step #3 (the last step).

5.4.2. Methods of the Signing Session

This group of methods implements a session that signs the provided data using previously enrolled token.

1. **createSignRequest**

   Performs Step #1 (create the request).

2. **updateSignRequest**

   Performs Step #2 (processing the response).

3. **finalizeSignRequest**

   Performs Step #3 (the last step).

5.4.3. Methods of the Refresh Session

This group of methods implements session that refreshes the signature token.

1. **createRefreshRequest**

   Performs Step #1 (create the request).

2. **updateRefreshRequest**

   Performs Step #2 (processing the response).

3. **finalizeRefreshRequest**

   Performs Step #3 (the last step).

5.4.4. Methods of the Delete Session

This group of methods implements session that deletes the signature token.

1. **createDeleteRequest**

   Performs Step #1 (create the request).

2. **updateDeleteRequest**

   Performs Step #2 (processing the response).

3. **finalizeDeleteRequest**

   Performs Step #3 (the last step).

5.5. Offline One-Time Password

The `DYOTP` class provides the following groups of methods:

- Create (enroll) token that is used for offline OTP calculation.
- Use the token to compute the offline OTP.
- Refresh the token. This functionality is optional.
- Delete the token.

5.5.1. Methods of the Enroll Session

This group of methods implements sessions that create tokens used in the subsequent data signing requests.

1. **createOfflineOTPToken**

   Create an offline OTP token.

2. **createEnrollmentRequest**

   Create an enrollment request.

3. **updateEnrollmentRequest**

   Handle the server response.

4. **finalizeEnrollmentRequest**

   Handling server response.

5.5.2. Methods of the OTP Computation

This method implements a session that signs the provided data using previously enrolled token.

1. **computeOTP**

   Perform mobile side OTP computation.

5.5.3. Methods of the Refresh Session

This group of methods implements a session that refreshes the signature token.

1. **createRefreshRequest**

   Create a refresh request.

2. **updateRefreshRequest**

   Handle the server response.

3. **finalizeRefreshWithContext**

   Handle the server response.

5.5.4. Sample Code

Sample code is provided in the package in the following location:

```
\samples\OfflineOTP
```

5.6. Encryption Protocol

CoT enables you to encrypt data on the client, start the decryption on the client, and then get the decrypted value on the smart proxy. The following methods are used for this protocol.

Methods of the Enroll Session

This group of methods implements sessions that create tokens used in the subsequent data signing requests.

1. createEnrollmentRequest
2. updateEnrollmentRequest
3. finalizeEnrollmentRequest

Methods of the Decryption Sessions

This group of methods implements a session that decrypts the provided data using previously enrolled token.

1. createDecryptRequest
2. updateDecryptRequest
3. finalizeDecryptRequest

Methods of the Refresh Session

This group of methods implements session that refreshes the signature token.

1. createRefreshRequest
2. updateRefreshRequest
3. finalizeRefreshWithContext

6. Appendix

6.1. One-Time Password (OTP)

CoT SDK supports the following OTP variants:

- Time-based OTP ("TOTP")
- HMAC-based OTP ("HOTP")

In the case of HOTP, the created password remains valid until the new HOTP is created.

A TOTP password is intended for short-time use – passwords expire every 30 seconds.

For example, if a TOTP is created on the 31st-second tick, it remains valid for another 29 seconds. Likewise, if a TOTP is set up on the 55th-second tick, it is valid for just 5 seconds.

Due to possible time glitches between two systems and message propagation delays, the submitted TOTP token is usually compared with the one that could be presented during `t-1, t, t+1` periods (previous, current, next).

6.1.1. Class Overview

```
com.dyadicsec.mobile.tokens.otp
java.lang.Object
    com.dyadicsec.mobile.DYSimpleBase
       com.dyadicsec.mobile.tokens.enc.DYOtp
```

`DYOtp` is a wrapper class for using the Unbound CoT one-time password functionality. It provides methods for the high-level operations: creating and deleting an OTP.

6.1.2. Create the OTP Token

```
public void createSignToken(
        String label,
        String username,
        DYCredentials credentials,
        Map<String, String> parameters,
        DYOTPTokenFactory.DYInitTokenListener listener)
```

Attention
This method deletes any other tokens of type `OTP` on the device.

Parameters:

- label - name assigned to the token
- username - name of the user
- credentials - credentials
- parameters - `null` – For HOTP variant. For additional options, see the note below.
- listener - the callback handler

To create a TOTP variant, assign to `parameters` the following value:

```
Map<String, String> parameters = new HashMap<>();
parameters.put(DYOTPTokenFactory.OTP_TOKEN_TYPE_PARAM, "TOTP");
```

6.1.3. Compute the OTP Token

Computes the next OTP.

```
public void computeOTP(
       DYCredentials credentials,
       IDYOTPToken.DYComputeOTPListener listener)
```

Parameters:

- credentials - Credentials
- listener - The callback handler.

6.2. Controlling Log Level

CoT SDK uses Android's `Logcat`. To change the log level in the runtime use:

```
DYLog.setLogLevel(level);
```

The `level` values match those of Android's `Logcat`:

> **Note**
> By default, the level of the log is set to `Warning`.

The available log-levels match those of Logcat:

```
DYLog.VERBOSE = 2;
DYLog.DEBUG = 3;
DYLog.INFO = 4;
DYLog.WARN = 5;   // Default
DYLog.ERROR = 6;
DYLog.ASSERT = 7;
```

Note

The order of verbosity, from terse to inflated, is: ERROR, WARN, INFO, DEBUG, VERBOSE. Logs are accessed using standard `Logcat` methods.

6.3. Return Values (Status Codes)

This section specifies status codes returned by the CoT SDK core package.

CoT SDK Return Values (Status Codes)

Value	Description
DY_SUCCEED	Operation succeeded.
DY_EFAIL	General error. The program encounters unexpected situation.
DY_EINVAL	Not valid argument.
DY_ENOENT	Not found error. Possible cause: • Trying to use a token before creating one. • Certificate file not found. • SDK failed to communicate with the Android Wear.
DY_ECOMM	Communication error. Possible causes: • URI of the CoT server is invalid. • Device communication issue. • Not a valid response from the server. • A communication error with Apple Watch.
DY_EDUP	This error code is reserved.
DY_EUNKNOWN	This error code is reserved.
DY_EBUSY	This error code is reserved.
DY_ECRYPT	Cryptography error. Indicates a mistake during generation or use of a key in the key store.
DY_ESIZE	This error code is reserved.
DY_EUSERABORT	The operation was canceled by the user. Possible causes: • User canceled the Touch ID signing operation. • User canceled Apple Watch approval operation.
DY_ETOUCHID_NOAVAIL	Touch ID is not available, or there is no fingerprint sensor. Possible causes:

Value	Description
	• The device does not support fingerprint authentication. • Error while checking the device fingerprint status.
DY_EINCORRECT_PIN	Incorrect PIN Possible causes: • Wrong PIN. • Fingerprint authentication failed. • Token with empty PIN.
DY_EKEYSYNC	Key sync error. Possible cause: Cryptographic error raised while trying to sync server and mobile keys/data.
DY_ESTATE_ CHANGED	The state was changed. Possible causes: • Fingerprint sensor was added/removed from the device. • Lock screen was disabled, re-create token (security issue).
DY_ETIMEOUT	Operation timeout reached. Possible causes: • Communication problem (server is not reachable). • The requested operation is taking more time than expected.
DY_ELOCKED	Communication with the server is blocked by the protocol. Possible cause: Too many incorrect retries (wrong PIN). According to the server policy, there are two options: • Wait a predefined time; the token is unlocked automatically. • The server locks the token forever. You must re-enroll the token.
DY_EPROTOCOL	Low-level cryptographic protocol error.
DY_EINVALIDSIG	The server certificate is invalid.
DY_EEXPIRED	Server key expired.
DY_EINIT	CoT SDK operation called before CoT SDK initialization.
DY_EOS	Android OS version is not valid.
DY_ETOUCHID_ PASSCODE_NOT_SET	Screen lock is not activated on the device. Possible cause: Using `DYFingerprintCredentials` on a device without enabling the screen lock. Suggestion: Enable the screen lock on the device and add at least one fingerprint before

Value	Description
	creating the token.
DY_EREFRESH_ REQUIRED	Mobile device exceeded the out-of-sync threshold for the token domain it is using. Possible cause: A deficiency in the connection to the server or in the finalization of the server response processing on the mobile device.