



Unbound Crypto-of-Things

Developer's Guide for iOS

Version 1.3

January 2019

Table of Contents

1. Revision History	1
2. Installation	2
2.1. System Requirements	2
2.2. Install CoT SDK	2
2.3. Add the Certificate	4
3. Introducing Common Concepts	6
3.1. Direct and Proxy-Based Deployment	6
3.2. User Credentials	7
3.2.1. PIN-Based Authentication	8
3.2.2. TouchId Authentication	8
3.2.3. LDAP Authentication	9
3.2.4. Void Authentication	10
4. Using Direct Access and Simple-Proxy	11
4.1. Initialization	11
4.1.1. Initialization in the Direct Connection Mode	11
4.1.2. Initialization in the Simple-Proxy Connection Mode	14
4.2. Password Protection	15
4.2.1. Class Overview	15
4.2.2. Sample Code	16
4.3. Digital Signature	17
4.3.1. Class Overview	17
4.3.2. Sample Code	19
4.4. Data Encryption	19
4.4.1. Sample Code using Direct Access	20
5. Using Smart-Proxy	21
5.1. Smart Entry Point Processing Paradigm	21
5.2. Password Protection	21
5.2.1. Password Enroll Method	22
5.2.2. Sample Code	24
5.3. Digital Signature	27

5.3.1. Methods of the Enroll Session	27
5.3.2. Methods of the Signing Session	29
5.3.3. Methods of the Refresh Session	30
5.3.4. Methods of the Delete Session	31
5.3.5. Sample Code	32
5.4. Offline One-Time Password	34
5.4.1. Methods of the Enroll Session	34
5.4.2. Methods of the OTP Computation	35
5.4.3. Methods of the Refresh Session	36
5.4.4. Sample Code	37
6. Appendix	38
6.1. One-Time Password (OTP)	38
6.1.1. Sample Code using Direct Access	38
6.2. Controlling Global Properties	39
6.2.1. Controlling Log Level	39
6.2.2. Controlling Timeouts	39
6.3. Return Values (Status Codes)	39

1. Revision History

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

Version	Date	Description
1.3	January 2019	Updated for version 1.3.
1.2	December 2018	Added a note about frameworks to Installation . Added section Offline One-Time Password .
1.2	August 2018	Added information about static frameworks to Installation .
1.2	June 2018	Added a note about the <i>ServerCertificate</i> parameter in section Digital Signature .
1.2	April 2018	Rebranded.
1.2.1706		Separated smart-proxy from the direct-access and simple proxy. New: smart proxy samples and methods to match the new implementation.
1.2.1704		New "Controlling Timeouts" Section. New "Setting URL Parameters" Section. New "LDAP-based Authentication" Section. 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

- Apple iPhone/iPad - iOS 8 and later
- Apple Watch – Watch OS 2.0 and later

2.2. Install CoT SDK

Obtain the Unbound SDK for iOS from Unbound.

Extract content from the package, noting the path to the folder where the files are stored. CoT SDK *.framework files are forwarded to the root directory.

Notes

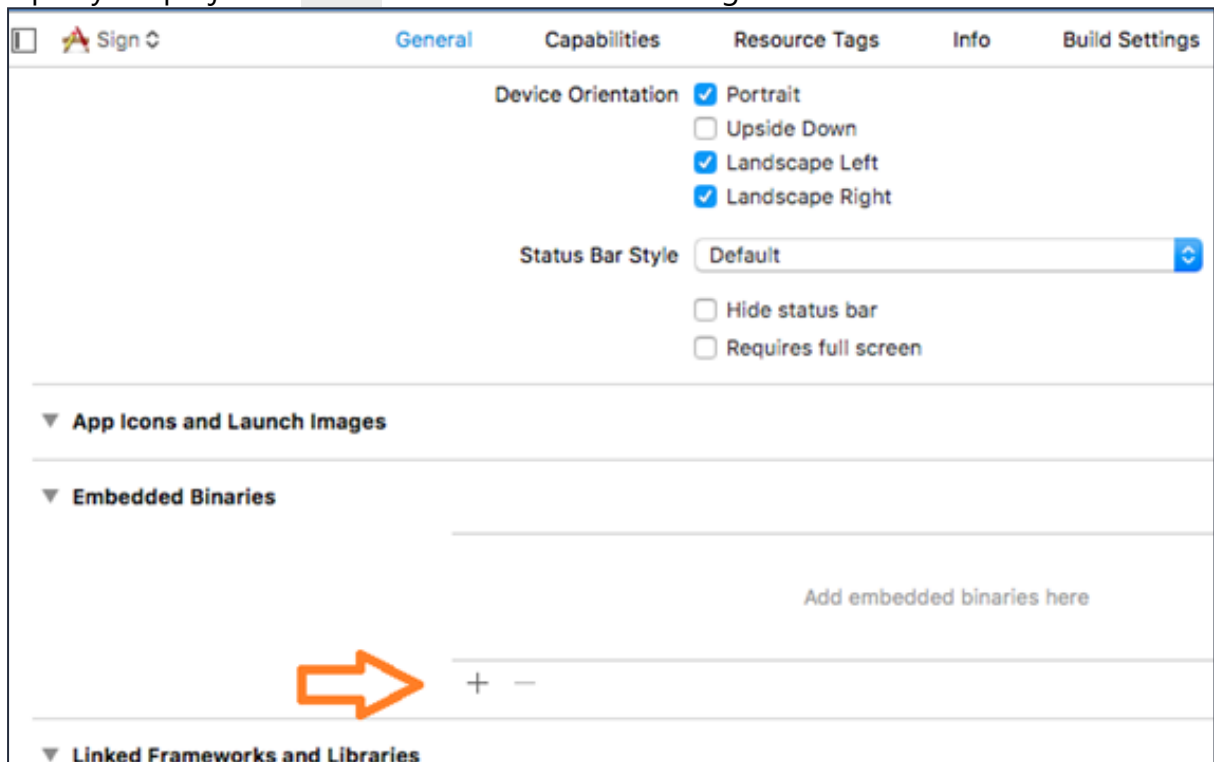
- CoT SDK framework modules allow you to hand-pick the minimal set that is required for your use case(s).
- Different files are required if you are using a **static** framework or a **dynamic** framework. Make sure that you have the correct package corresponding to your framework type.

Framework File	Description
iOS Apps	
DYMobileCore.framework	This module is required. Make sure that you use the dynamic or static framework, depending on your implementation.
DYMobilePWD.framework	Enables use of password tokens.
DYMobileSign.framework	Enables use of signing tokens. Make sure that you use the dynamic or static framework, depending on your implementation.
DYMobileEnc.framework	Enables use of encryption tokens.
DYMobileOTP.framework	Enables use of OTP tokens.
openssl.framework	Required when using the static framework.
iOS Watch Apps	
DYMobileCoreWatchOS.framework	This module is required.
DYMobileOTPWatchOS.framework	Enables use of OTP tokens.

Other entries of the package are organized in the subfolders of the current directory as follows:

Subfolder	Description
without simulator	A folder with CoT SDK Framework modules for iOS stripped from simulator objects (an iTunes-friendly version of the CoT SDK).
samples	A folder with code samples for the use cases.
doc	Start from the <i>index.html</i> file.

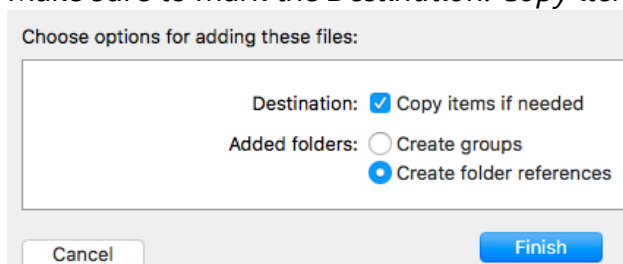
Open your project in Xcode and click the **General** tag.







Add the required file:

- Dynamic framework: Click the "+" icon in the *Embedded Binaries* section and select the *Core* framework file.
- Static framework: Click the "+" icon in the *Linked Frameworks and Libraries* section and select the *Core* framework file.

Make sure to mark the *Destination: Copy items if needed* and click **Finish**.



Repeat the two previous steps choosing the framework files that support your use case. For example, the following set is used in dynamic signing applications.

▼ Embedded Binaries		
	 DYMobileSign.framework	
	 DYMobileCore.framework	
	+ -	
▼ Linked Frameworks and Libraries		
	Name	Status
	 DYMobileSign.framework	Required ↕
	 DYMobileCore.framework	Required ↕

Note

If you are implementing the static framework, you must also add `-ObjC -lstdc++` to the *Other Linker Flags* in the **Build Settings** tab.

Note

If you switch from the embedded to static framework, you must remove `DYMobileCore.framework` and `DYMobileSign.framework` from the **Embedded Binaries** section.

2.3. Add the Certificate

The CoT SDK uses a public key to encrypt the data it is sending to the CoT server. This key and the corresponding signature that certifies the sender of the key and its integrity are forwarded to the mobile device by the UKC server. To validate the signature, the CoT SDK uses a certificate that is hard-coded in the application.

CoT SDK requires a "DER-encoded" certificate.

If you received Certificate in standard **.pem* format (e.g., *server-certificate.pem*) run the following command to convert it into the "DER" format:


```
openssl x509 -inform PEM -outform DER \
-in server-certificate.pem -out server-certificate.cer
```

Drag the *.cer* file into `Xcode` source tree. The folding dialog pop up:

Choose options for adding these files:

Destination: ☒ Copy items if needed

Added folders: ☐ Create groups
☒ Create folder references

Add to targets: ☒  Sign

Cancel Finish

Select *Copy items if needed* and *Add to targets*. Check that the target indicates your project (in our case the name of the project is Sign).

Click **Finish**.

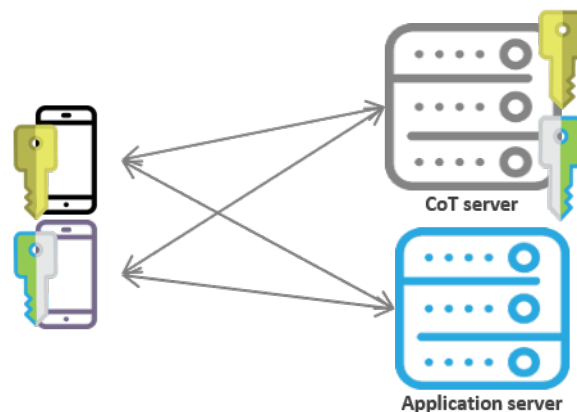
3. Introducing Common Concepts

3.1. Direct and Proxy-Based Deployment

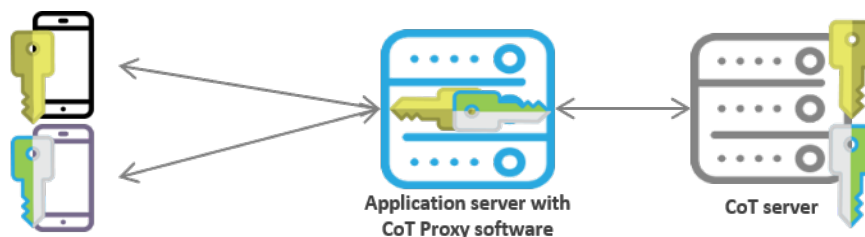
A mobile app can address its CoT server in two ways:

- Directly, by specifying its URL (or an endpoint on a standard reverse HTTP proxy).
- Via an **CoT Proxy** that is deployed on the app server.

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

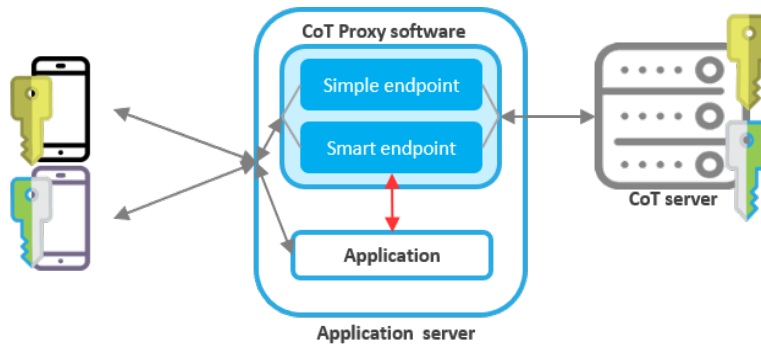


Indirect access - The mobile application addresses the CoT server via CoT proxy that runs on an application server.



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 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 has been performed by the app server.

These three connection methods (direct, simple proxy, 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.

3.2. 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. 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

To bind PIN to a new token, use `DYPINCredentials` object and apply `initWithPIN` method when creating the token.

The following snippet shows binding between PIN-based authentication and token (`sharedDYMobile`) that is utilized in the subsequent signing procedures.

```
[[DYMobile sharedDYMobile]
createSignTokenForUsername:@"User Name"
    withCredentials:[[DYPINCredentials alloc] initWithPIN:@"1234"]
andProtectionParameters:nil //RSA
    completionHandler:^(DYToken * _Nonnull token, NSError * _Nullable error)
{
    // code deleted
}
]
```

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. TouchId Authentication

Note

You can use the following helper function to check if Touch method is provided on the device

```
(BOOL) canAuthenticateByTouchId: (NSError **_Nullable) error
```

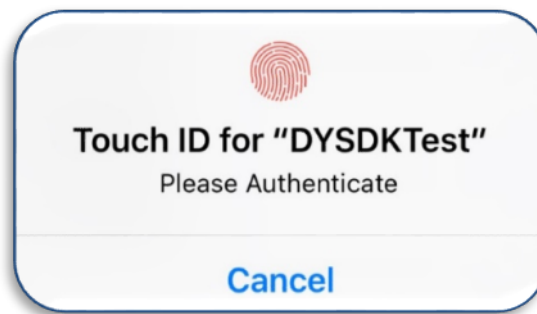
To bind `TouchID` to a new token, use `DYTouchIDCredentials` object and apply `initWithTouchIDMessage` method when creating the token.

The following snippet shows binding between Touch ID-based authentication and token (`sharedDYMobile`) that is utilized in the subsequent signing purposes.

```
[[DYMobile sharedDYMobile]
createSignTokenForUsername:@"User Name"
    withCredentials:[[DYTouchIDCredentials alloc]
```

```
initWithTouchIDMessage:@"Please Authenticate"]
andProtectionParameters:nil // RSA
completionHandler:^(DYToken * _Nonnull token, NSError * _Nullable error)
{
}
];
```

The `initWithTouchIDMessage` method takes as a parameter a string that is displayed in the Generic iOS Message Box that prompts to perform the Touch ID identification.



Referring to the previous snippet, "Please Authenticate" line appears in the generic iOS Message Box that prompts the user to perform touch identification.

3.2.3. LDAP Authentication

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. Please consult with the Admin of the CoT Server which of the following formats should be used in your application:

Table 1: Formats of Username for Use with LDAP

Format	Comment
username	
domain\username	Also called NetbiosName\sAMAccountName
username@domain.com	Also called UPN
(cn=some,cn=ou,dc=domain,dc=com)	Also called "Distinguished Name"

To bind PIN to a new token, use `DYPINCredentials` object and apply `initWithPIN` method when creating the token.

The following snippet shows binding between PIN-based authentication and token (`sharedDYMobile`) that is utilized in the subsequent signing procedures.

```
[[DYMobile sharedDYMobile]
createSignTokenForUsername:@"User Name"
withCredentials:[[DYLDAPCredentials alloc]
initWithUserName:@"User1"
andPassword:@"Password1"]
andProtectionParameters:nil //RSA
```

```
completionHandler:^(DYToken * _Nonnull token, NSError * _Nullable error)
{
}
]
```

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.

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. Allocate singleton of the CoT SDK - `DYMobile sharedDYMobile`.
2. Initialize it using `initDYMobileWithURL` while specifying the URL of the CoT server.

```
-(BOOL) initDYMobileWithURL: (NSURL *) url
        domain: (NSString* _Nullable) domain
        deviceUID: (NSString *) deviceId
        certificateNamed: (NSString *) certificateName
        urlParams: (NSDictionary *_Nullable) urlParams
        error: (NSError **_Nullable) error
```

Parameters of `initDYMobileWithURL`:

- `url` - URL of the CoT server.
- `domain` - Name of the token domain on CoT Server assigned to handle your application. If omitted, the default token domain (DEFAULT_DOMAIN) is used.
- `urlParams` - May be omitted. Else, it is a list of key-value pairs in the format of `key1=val1%key2=val2`, etc.
- `deviceUID` - The UID of the device.
- `certificateNamed` - CA Certificate file name.

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 <code>USE_HEADERS</code> in the <code>urlParams</code> 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 <code>Authorization</code> 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:

```
[[DYMobile sharedDYMobile]
initDYMobileWithURL:[NSURL URLWithString: @"REPLACE WITH the CoT SERVER URL"]
    domain:nil
    deviceUID:[UIDevice currentDevice].identifierForVendor.UUIDString
    certificateNamed:@"REPLACE WITH YOUR CERTIFICATE.cer"
    urlParams:[NSDictionary dictionaryWithObjectsAndKeys:
        @"0a49b4a4-f824-46aa-b148-647a8e2e1d8a", @"client_id",
        @"pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM", @"client_secret",
        nil]
    error:&error];
```

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:

```
[[DYMobile sharedDYMobile]
initWithURL:[NSURL URLWithString:@"REPLACE WITH YOUR SERVER URL"]
domain:nil
deviceUID:[UIDevice currentDevice].identifierForVendor.UUIDString
certificateNamed:@"REPLACE WITH YOUR CERTIFICATE.cer"
urlParams:[NSDictionary dictionaryWithObjectsAndKeys:
@"0a49b4a4-f824-46aa-b148-647a8e2e1d8a", @"client_id",
@"pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM7", @"client_secret",
@"PLAIN", @"USE_HEADERS", nil]
error:&error];
```

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: pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM7tN5fA8o0
```

4.1.1.4. Base-Access Credentials in the HTTP Header

The method:

```
[[DYMobile sharedDYMobile]
initWithURL:[NSURL URLWithString:@"REPLACE WITH YOUR SERVER URL"]
domain:nil
deviceUID:[UIDevice currentDevice].identifierForVendor.UUIDString
certificateNamed:@"REPLACE WITH YOUR CERTIFICATE.cer"
urlParams:[NSDictionary dictionaryWithObjectsAndKeys:
@"0a49b4a4-f824-46aa-b148-647a8e2e1d8a", @"client_id",
@"pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM7", @"client_secret",
@"BASIC_AUTHENTICATION", @"USE_HEADERS", nil]
error:&error];
```

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: pK1eL6iQ2yU3oH2aJ5tB6uC1iY5iU0kM4nY3pR2nM7tN5fA8o0
```

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.

4.1.2. Initialization in the Simple-Proxy Connection Mode

Perform the following steps:

1. Create proxy – an object of the ProxyComm that is a sample implementation of DYComm class.
2. Allocate singleton of the CoT SDK - DYMobile sharedDYMobile.
3. Initialize it using initDYMobileWithComm while specifying the URL of the CoT server.

```
ProxyComm* proxy = [[ProxyComm alloc] init];

// initialize CoT SDK
[[DYMobile sharedDYMobile]
 initDYMobileWithComm:proxy
 domain:nil
 deviceUID:[UIDevice currentDevice].identifierForVendor.UUIDString
 certificateNamed:@"REPLACE WITH YOUR CERTIFICATE.cer"
 error:&error];
```

Parameters:

- 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 CoT Server assigned to handle your application. If omitted, the default token domain (DEFAULT_DOMAIN) is used.
- deviceUID - The UID of the device
- certificateNamed - CoT server certificate received from CA

ProxyComm

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

- The URL of the CoT Proxy (for example, the app server URL).
- The single endpoint that provides a proxy function.
- The SendRequest method that delivers the request and handles the response.

```
// ProxyComm.m
#import "ProxyComm.h"
#define PROXY_SERVER_URL @"URL of the App Proxy"
@implementation ProxyComm
-(void)sendRequest:(NSString*)operationID
             domain:(NSString*)domain
           withData:(id)request
completionHandler:(void (^)(id response, NSError* error))handler {
    NSError* error;
```

```

NSURL* url = [NSURL URLWithString:PROXY_SERVER_URL];
NSURL* urlPath = [url URLByAppendingPathComponent:@"/api/proxy"];

// Building proxy request:
//   add operation ID - mandatory
//   add domain
//   format it into JSON
//   Transmit the request to the proxy server

```

Parameters of `SendRequest`

- `operationID` - An operation that is performed by the CoT Server
- `domain` - Name of the token domain assigned to this application
- `request` - Request in json format
- `completionHandler` - Call-back procedure that handles the response

4.2. Password Protection

4.2.1. Class Overview

The `DYMobile+PWD` is a wrapper class for storing and retrieving a single password.

```

// DYMobile+PWD.h
// DYMobilePWD

```

4.2.1.1. `protectPassword`

```

-(void)protectPassword:(NSString*)password
    forUsername:(NSString*)username
    withLabel:(NSString*)label
    withCredentials:(DYCredentials*)credentials
    authenticationToken:(BOOL)authenticationToken
    andProtectionParameters:(NSDictionary *_Nullable)parameters
    completionHandler:(void (^)(DYToken* token,
        NSError *_Nullable error))handler;

```

Important

This method deletes any other tokens of type Password on the device (e.g., previously defined password becomes invalid).

Parameters:

- `password` - The password string - provided "as-is".
- `username` - Name of the user.
- `label` - Label assigned to the token.
- `credentials` - Credentials.

- authenticationToken -
NO – password can be reassembled from its shares further decryption.
YES- reassembly of the password shall require further decryption of the share stored on CoT server.

Note

This option assumes that the CoT server has been configured to encrypt its token-share using the public key provided by the organization that uses this application.

- parameters - nil – the default (not PQC) encryption or specification of the PQC encryption (see the note below).
- handler - The callback handler.

Note

To use PQC-grade encryption of the password, assign the following to the parameters:

[[NSDictionary alloc] initWithObjectsAndKeys:

```
[NSNumber numberWithInt:YES],
@"PQC",
nil]
```

4.2.1.2. retrievePasswordWithCredentials

```
-(void)retrievePasswordWithCredentials:(DYCredentials*)credentials
completionHandler:(void (^)(NSString* password,
NSString* authToken,
NSError *error))handler;
```

Parameters:

- credentials - Credentials.
- handler - 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

Init creates a singleton of the CoT SDK. See [Initialization](#).

4.2.2.2. Enrollment of the Password (Password Protection)

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

```
[[DYMobile sharedDYMobile]
    protectPassword:@"password"
    forUsername:@"username"
    withCredentials:[[DYPINCredentials alloc] initWithPIN:@"1234"]
    authenticationToken:NO
    andProtectionParameters:nil // standard (not PQC) encryption
    completionHandler:^(DYToken* token, NSError* error) {
        if (error!=nil) {
            // code deleted
        }
    }
    }];
```

4.2.2.3. Retrieval of the Password

The following sample of code triggers PIN-based authentication and, after verification proceeds to retrieve the password.

```
[[DYMobile sharedDYMobile]
    retrievePasswordWithCredentials:[[DYPINCredentials alloc] initWithPIN:@"1234"]
    completionHandler:^(NSString *password,
        NSString* authTicket,
        NSError* error) {
        if (error!=nil) {
            // code deleted
        }
    }
    }];
```

4.3. Digital Signature

4.3.1. Class Overview

The DYMobile+Sign 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.

```
// DYMobile+Sign.h
// DYMobileSign
```

4.3.1.1. createSignTokenForUsername

Important

This method deletes any other tokens of type Sign on the device

```
-(void)createSignTokenForUsername:(NSString*)username
withLabel:(NSString*)label
withCredentials:(DYCredentials*)credentials
```

```
andProtectionParameters:(NSDictionary *_Nullable)parameters
completionHandler:(void (^_Nullable)(DYToken* token, NSError* _Nullable
error))handler;
```

Parameters:

- username - Name of the user.
- label - Label assigned to the token.
- credentials - Credentials.
- parameters - `nil` – for RSA-based signing. For the additional options, see the note below.
- handler - The callback handler.

Note

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

```
[[NSDictionary alloc] initWithObjectsAndKeys: @"ECDSA", @"type",nil];
```

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. signData

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

```
-(void)signData:(NSData*)data
withCredentials:(DYCredentials*)credentials
hashAlgorithm:(DYHashAlgorithms)hashAlgorithm
completionHandler:(void (^)(NSData* signature, NSError* error))handler;
```

Parameters:

- data - The byte array to sign.
- credentials - Credentials
- hashAlgorithm - Optional parameter. If omitted: – the default is SHA256. Other values: SHA1, SHA256, SHA384, SHA512.
- handler - The callback handler.

4.3.2. Sample Code

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

4.3.2.1. Init

Init creates a singleton of the CoT SDK. See [Initialization](#).

4.3.2.2. Creating Signature Token

```
-(void)enrollWithPIN:(BOOL)isECDSA {
    NSDictionary* params = nil;
    if (isECDSA) {
        params = [[NSDictionary alloc] initWithObjectsAndKeys:@"ECDSA", @"type", nil];
    }
    NSString* pin = pinTextField.text;
    [[DYMobile sharedDYMobile]
    createSignTokenForUsername:@"User Name"
    withCredentials:[[DYPINCredentials alloc] initWithPIN:pin]
    andProtectionParameters:params
    completionHandler:^(DYToken * _Nonnull token,
    NSError * _Nullable error) {
        ;
    }
    }];
```

4.3.2.3. Signing

```
-(void)signWithPIN:(NSString*)plainText{
    NSData* data = [plainText dataUsingEncoding:NSUTF8StringEncoding];
    NSString* pin = pinTextField.text;
    [[DYMobile sharedDYMobile] signData:data
    withCredentials:[[DYPINCredentials alloc] initWithPIN:pin]
    completionHandler:^(NSData * _Nonnull signature,
    NSError * _Nullable error) {
        if (error!=nil) {
            NSLog(@"Error signing: %@ (%ld)",
            error.localizedDescription,
            (long)error.code);
        } else {
            computedSignature = signature;
        }
    }
    }];
```

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. Sample Code using Direct Access

4.4.1.1. Creating an Encryption Token

```
[[DYMobile sharedDYMobile]
createEncryptionTokenForUsername:@"username"
withCredentials:[[DYPINCredentials alloc] initWithPIN:pin]
andProtectionParameters:nil // ECIES with P256
completionHandler:^(DYToken *token, NSError* error) {
if (error!=nil) {
    // code deleted
}
}];
```

4.4.1.2. Using the Token for Encryption

Note

Authentication is not required

```
NSString *str = @"plain data";
NSData *data = [str dataUsingEncoding:NSUTF8StringEncoding];
NSError* error;
NSData* encData = [[DYMobile sharedDYMobile] encrypt:data error:&error];
if (error!=nil) {
    // code deleted
}
```

4.4.1.3. Using the Token for Decryption

Note

Authentication is required

```
[DYMobile sharedDYMobile] decrypt:encData
withCredentials:[[DYPINCredentials alloc] initWithPIN:pin]
completionHandler:^(NSData *plainData, NSError* error) {
if (error!=nil) {
    // code deleted
}
}];
```

5. Using Smart-Proxy

Disclaimer

Communication with the App Server is implemented in the sample code for demonstration and POC purpose 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. 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.

5.2.1. Password Enroll Method

5.2.1.1. protectPassword

Stores the password in CoT. Uses simple endpoint. Hence it is implemented in one step.

```
-(void)protectPassword:(NSString*)password
forUsername:(NSString*)username
withCredentials:(DYCredentials*)credentials
authenticationToken:(BOOL)authenticationToken
completionHandler:(void (^)(NSError* error))handler;
```

Parameters:

- password - The password string - provided "as-is".
- username - Name of the user.
- credentials - Credentials.
- authenticationToken - NO – password can be reassembled from its shares further decryption.

Note

Other options are not supported by smart endpoint

- handler - Callback procedure. Called by this method when it completes the send-receive-handle cycle.

5.2.1.2. 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.

5.2.1.3. createPasswordRetrieveRequestWithCredentials

It creates:

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

```
-(void)createPasswordRetrieveRequestWithCredentials:(DYCredentials*)credentials
completionHandler:(
void (^)(NSDictionary* request,
NSData* context,
NSError* error)
)handler;
```

Parameters:

- `credentials` - Credentials.
- `handler` - A call-back procedure called by this method once it completes its part of creating the request.

Parameters:

- `request` - Payload to be delivered to the server.
- `context` - CoT SDK data that application must pass to the next step.
- `error` - Indication whether the above data is valid or not.

5.2.1.4. updatePasswordRetrieveRequestWithResponse

Uses the response received via the call back of the of the `createPasswordRetrieveRequest` to perform the update step of the crypto operation.

```
-(void)updatePasswordRetrieveRequestWithResponse:(NSDictionary *)response
credentials:(DYCredentials*)credentials
additionalRequest:(NSDictionary*_Nullable* _Nullable)additionalRequest
context:(NSData** _Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- `response` - The content of the response from the server.
- `credentials` - Credentials.
- `additionalRequest` - A reference to an additional request object. Set by the CoT SDK.
`nil` indicates that the CoT SDK has no more requests.
Anything else indicates the additional request that must be performed by the application logic.

- context - CoT SDK data (provided in Step #1 by the callback of the `createPasswordRetrieveRequestWithCredentials`).
- error - Error reference object.

5.2.1.5. `finalizePasswordRetrieveWithContext`

Finalizes (completes) password retrieval operation.

Note

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

```
-(BOOL)finalizePasswordRetrieveWithContext:(NSData*)context
```

```
error:(NSError** _Nullable) error;
```

Parameters:

- context - Operation context (provided by the callback of the `createPasswordRetrieveRequestWithCredentials`).
- error - Error reference object.

5.2.2. Sample Code

Sample code for password protection demonstrates the use of mix-mode: password protection is using a simple proxy, while the password retrieval uses a smart proxy.

As such, password protection uses the sample `ProxyClass` to implement communication with CoT server, while password retrieval depends on the application itself to communicate with the server.

5.2.2.1. Init

Init creates `DYMobile` `sharedDYMobile` singleton and binds it with the `ProxyComm` object.

Note

As noted above, `ProxyComm` is used only for password protection.

The steps are identical to the one described in the Simple Proxy Initialization:

1. Create proxy – an object of the `ProxyComm` that is a sample implementation of the `DYComm` class.
2. Allocate singleton of the CoT SDK - `DYMobile` `sharedDYMobile`.
3. Initialize it using `initDYMobileWithComm` while specifying the URL of the CoT server.

```
ProxyComm* proxy = [[ProxyComm alloc] init];
// initialize the CoT SDK with the "proxy" object of DYComm
```

```
[[DYMobile sharedDYMobile]
initDYMobileWithComm:proxy
domain:nil
deviceUID:[UIDevice currentDevice].identifierForVendor.UUIDString
certificateNamed:@"REPLACE WITH YOUR CERTIFICATE.cer"
error:&error];
```

Parameters of the `initDYMobileWithComm`:

- 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 CoT server assigned to handle your application. If omitted, the default token domain (DEFAULT_DOMAIN) is used.
- deviceUID - The UID of the device.
- certificateNamed - CA Certificate file name.

5.2.2.2. ProxyComm

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

- The URL of the CoT Proxy (for example, the app server URL).
- The single endpoint that provides proxy function.
- The `SendRequest` method that delivers the request and handles the response.

```
// ProxyComm.m
#import "ProxyComm.h"
#define PROXY_SERVER_URL @"REPLACE_WITH_the_EKP_PROXY_URL"

@implementation ProxyComm
-(void)sendRequest:(NSString*)operationID
              domain:(NSString*)domain
            withData:(id)request
completionHandler:(void (^)(id response, NSError* error))handler {
    NSError* error;

    NSURL* url = [NSURL URLWithString:PROXY_SERVER_URL];
    NSURL* urlPath = [url URLByAppendingPathComponent:@"/api/proxy"];
    // Building proxy request.
    // add operation ID - mandatory
    // add domain
    // format it into JSON
    // Transmit the request to the proxy server
```

Parameters of the `SendRequest`

- operationID - An operation that is performed by the CoT server.
- domain - Name of the token domain assigned to this application.
- request - Request in *json* format.
- completionHandler - Call-back procedure that handles the response.

5.2.2.3. Enrollment of the Password (Password Protection)

Similar to the Init, the enrollment of the new password is identical to the one used in the simple proxy case.

5.2.2.4. Retrieval of the Password

The password is reassembled on the application server. Therefore, the request to retrieve the password includes the password-share saved on the mobile device and additional crypto-info. The application performs the following steps:

1. Ask the CoT SDK to prepare the request data:

```
[[DYMobile sharedDYMobile]
createPasswordRetrieveRequestWithCredentials:credentials
completionHandler:^(
    NSDictionary *request,
    NSData *context,
    NSError *error
)
```

This is async-call. Once the CoT SDK prepares the request, it calls the provided `completionHandler` passing it the `request`. Go to Step 2.

2. Add the endpoint on the app server (`api/proxy/password/retrieve`) and send.

Note

In the sample code we duplicate the send method implemented by the `ProxyComm`.

```
[self sendRequest:proxyRequest
url:@"api/proxy/password/retrieve"
completionHandler:^(
    id _Nullable response,
    NSError * _Nullable error
)
```

This is async-call. Once `ProxyComm` receives the response, it calls the provided `completionHandler` passing it the response. Go to Step 3.

3. Call CoT SDK to process the response.

```
[[DYMobile sharedDYMobile]
updatePasswordRetrieveRequestWithResponse:response
credentials:credentials
```

```
additionalRequest:&additionalRequest
context:&blockContext
error:&err
];
```

If everything is OK and no new requests are required by the CoT SDK, then proceed to Step 4.

4. Call CoT SDK to finalize the processing of the password retrieval.

```
[[DYMobile sharedDYMobile]
finalizePasswordRetrieveWithContext:context
error:&err];
```

5.3. 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.

5.3.1. Methods of the Enroll Session

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

5.3.1.1. `createSignTokenEnrollmentRequestForUsername`

Performs Step #1 (create the request).

```
-(NSDictionary*)
createSignTokenEnrollmentRequestForUsername:(NSString*)username
withLabel:(NSString*)label
withCredentials:(DYNonBlockingCredentials*)credentials
andProtectionParameters:(NSDictionary *_Nullable)parameters
context:(NSData *_Nullable *_Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- username - Name of the user.
- label - Label assigned to the token.
- credentials - Credentials.
- parameters - `nil` – for RSA-based signing. (It is the only supported type of signature encryption in the smart proxy case).
- context - Operation context reference.
- error - Error reference.

5.3.1.2. updateSignTokenEnrollmentWithResponse

Performs Step #2 (processing the response).

```
-(void)
updateSignTokenEnrollmentWithResponse:(NSDictionary *)response
credentials:(DYNonBlockingCredentials*)credentials
additionalRequest:(NSDictionary*_Nullable*_Nullable)additionalRequest
context:(NSData*_Nullable*_Nullable)context
error:(NSError**_Nullable)error;
```

Parameters:

- response - Response from the server.
- credentials - Credentials.
- additionalRequest - A reference to additional request object. Set by the CoT SDK. `nil` indicates that the CoT SDK has no more requests. Anything else indicates the additional request that must be performed by the application logic.
- context - Operation context reference.
- error - Error reference.

5.3.1.3. finalizeSignTokenEnrollmentWithContext

Performs Step #3 (the last step).

```
-(BOOL)
finalizeSignTokenEnrollmentWithContext:(NSData*)context
error:(NSError**_Nullable)error;
```

Parameters:

- context - Operation context reference.
- error - Error reference.

5.3.2. Methods of the Signing Session

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

5.3.2.1. createSignRequestWithCredentials

Performs Step #1 (create the request).

```
-(NSDictionary*)
createSignRequestWithCredentials:(DYNonBlockingCredentials*)credentials
dataToSign:(NSData*)dataToSign
context:(NSData*_Nullable*_Nullable)context
error:(NSError**_Nullable)error;
```

Parameters:

- credentials - Credentials.
- dataToSign - Data for signing.
- context - Operation context reference.
- error - Error reference.

5.3.2.2. updateSignRequestWithResponse

Performs Step #2 (processing the response).

```
-(void)
updateSignRequestWithResponse:(NSDictionary *)response
credentials:(DYNonBlockingCredentials*)credentials
additionalRequest:(NSDictionary*_Nullable*_Nullable)additionalRequest
context:(NSData*_Nullable*_Nullable)context
error:(NSError**_Nullable)error;
```

Parameters

- response - Response from the server.
- credentials - Credentials.
- additionalRequest - A reference to additional request object. Set by the CoT SDK. `nil` indicates that CoT SDK has no more requests. Anything else indicates the additional request that must be performed by the application logic.
- context - Operation context reference.
- error - Error reference.

5.3.2.3. finalizeSignWithContext

Performs Step #3 (the last step).


```
-(BOOL)
finalizeSignWithContext:(NSData*)context
error:(NSError** _Nullable) error;
```

Parameters:

- context - Operation context reference.
- error - Error reference.

5.3.3. Methods of the Refresh Session

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

5.3.3.1. createRefreshSignTokenRequestWithContext

Performs Step #1 (create the request).

```
-(NSDictionary*)
createRefreshSignTokenRequestWithContext:(NSData*_Nullable* _Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- context - Operation context reference.
- error - Error reference.

5.3.3.2. updateRefreshSignTokenWithResponse

Performs Step #2 (processing the response).

```
-(void)
updateRefreshSignTokenWithResponse:(NSDictionary *)response
additionalRequest:(NSDictionary*_Nullable* _Nullable)additionalRequest
context:(NSData*_Nullable* _Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- response - Response from the server.
- additionalRequest - A reference to an additional request object.
nil indicates that this is the last request.
- context - Operation context reference.
- error - Error reference.

5.3.3.3. finalizeRefreshSignTokenWithContext

Performs Step #3 (the last step).

```
-(BOOL)
finalizeRefreshSignTokenWithContext:(NSData*)context
error:(NSError** _Nullable) error;
```

Parameters:

- context = Operation context reference.
- error - Error reference.

5.3.4. Methods of the Delete Session

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

5.3.4.1. createDeleteSignTokenRequestWithContext

Performs Step #1 (create the request).

```
-(NSDictionary*)
createDeleteSignTokenRequestWithContext:(NSData*_Nullable* _Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- context - Response from the server.
- error - Error reference.

5.3.4.2. updateDeleteSignTokenWithResponse

Performs Step #2 (processing the response).

```
-(void)
updateDeleteSignTokenWithResponse:(NSDictionary *)response
context:(NSData*_Nullable* _Nullable)context
error:(NSError** _Nullable)error;
```

Parameters:

- response - Response from the server.
- context - Operation context reference.
- error - Error reference.

5.3.4.3. finalizeDeleteSignTokenWithContext

Performs Step #3 (the last step).

```
-(BOOL)
finalizeDeleteSignTokenWithContext:(NSData*)context
error:(NSError** _Nullable) error;
```

Parameters:

- context - Operation context reference.
- error - Error reference.

5.3.5. Sample Code

Note

We document the enrollment sample. The rest of methods (signing, refreshing, deleting) follow the same pattern.

5.3.5.1. Enroll Sample (createSignToken)

This sample code performs the following steps:

1. Update the server-info before performing enrollment (must be the 1st call the server).

```
NSDictionary* serverInfoResponse = [self loadServerInfo];
[[DYMobile sharedDYMobile]
updateServerInfoFromServerResponse:serverInfoResponse
```

2. create credentials object to be used during the enrollment.

```
DYPINCredentials* credentials = [[DYPINCredentials alloc] initWithPIN:@"1234"];
```

3. Declare the context that is used across 3-step session

```
NSData* context;
```

- a. Step 1: Call CoT SDK to create the `enrollmentRequest` and to initialize the context

```
NSDictionary* enrollmentRequest = [[DYMobile sharedDYMobile]
createSignTokenEnrollmentRequestForUsername:@"username"
withLabel:@"label"
andProtectionParameters:nil
context:&context
error:&error];
```

Call helper function to send the `enrollmentRequest`

```
NSDictionary*
enrollmentResponse = [self sendRequest:[self
createSmartProxyRequestFromRequest:
[[NSDictionary alloc]
initWithObjectsAndKeys:enrollmentRequest,@"requestData", nil]
proxyPayload:nil]
url:@"api/proxy/rsa/enroll"]];
```

- b. Step 2: Call CoT SDK to process the `enrollmentResponse` in the context set by the 1st step.

```
NSDictionary* additionalRequest;
[[DYMobile sharedDYMobile]
updateSignTokenEnrollmentWithResponse:enrollmentResponse
credentials:credentials
additionalRequest:&additionalRequest
```

```
context:&context
error:&error];
```

Examine the content of the `additionalRequest`. If none, proceed to the last step.

- c. Step 3: Finalize the session (context):

```
BOOL b = [[DYMobile sharedDYMobile]
finalizeSignTokenEnrollmentWithContext:context
error:&error];
```

5.3.5.2. Helper Methods

Sample Code for sending request for server info (via simple endpoint)

```
-(NSDictionary*)
loadServerInfo {
return [self sendRequest:[self createSimpleProxyRequestFromRequest:[NSDictionary
alloc] init] operation:@"info"]
url:@" /api/proxy"];
}
```

Sample code for assembling request designated to simple endpoint

```
-(NSDictionary*)
createSimpleProxyRequestFromRequest:(NSDictionary*)request
operation:(NSString*)operation
{
NSMutableDictionary* proxyRequest = [[NSMutableDictionary alloc] init];
[proxyRequest setObject:request forKey:@"requestData"];
[proxyRequest setObject:operation forKey:@"operationID"];
// add domain to the proxy request
if (domain!=nil) [proxyRequest setObject:domain forKey:@"domain"];
return proxyRequest;
}
```

Sample code for assembling request designated to smart endpoint

```
-(NSDictionary*)
createSmartProxyRequestFromRequest:(NSDictionary*)request
proxyPayload:(NSDictionary*)proxyPayload
{
NSMutableDictionary* proxyRequest = [[NSMutableDictionary alloc]
initWithDictionary:request];
if (proxyPayload!=nil) [proxyRequest setObject:proxyPayload forKey:@"proxyPayload"];
// add domain to the proxy request
if (domain!=nil) [proxyRequest setObject:domain forKey:@"domain"];
return proxyRequest;
}
```

5.4. Offline One-Time Password

The `DYOfflineOTP` 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.

5.4.1. Methods of the Enroll Session

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

5.4.1.1. `createOfflineOTPTokenWithLabel`

Create an offline OTP token.

```
NSError* error;
DYOfflineOTPToken* token = [[DYMobile sharedDYMobile].otpTokenFactory
createOfflineOTPTokenWithLabel:@"Label"
username:@"User Name"
withCredentials:credentials
parameters:nil
error:&error];
```

Parameters:

- Label – Token label (used for audit).
- username - Name of the user.
- withCredentials - Credentials.
- parameters - `nil`.
- error - Error reference.

5.4.1.2. `createEnrollmentRequestWithCredentials`

Create an enrollment request.

```
NSData* context;
NSDictionary* enrollmentRequest = [token
createEnrollmentRequestWithCredentials:credentials
enrollmentParameters:nil
context:&context
error:&error];
```

Parameters:

- `credentials` - Reference to *OfflineOTPCredentials* with a 16 byte (32 hex chars) key.
- `enrollmentParameters` - `nil`.
- `context` - Operation context reference.
- `error` - Error reference.

5.4.1.3. `updateEnrollmentWithResponse`

Handle the server response.

```
[token updateEnrollmentWithResponse:enrollmentResponse
credentials:credentials
additionalRequest:&additionalRequest
context:&context
error:&error];
```

Parameters:

- `credentials` - Reference to *OfflineOTPCredentials* with a 16 byte (32 hex chars) key.
- `additionalRequest` - (optional) Reference to operation additional request.
- `context` - Operation context reference.
- `error` - Error reference.

5.4.1.4. `finalizeEnrollmentWithContext`

Handling server response

```
BOOL b = [token finalizeEnrollmentWithContext:context
error:&error];
```

Parameters:

- `error` - Error reference.

5.4.2. Methods of the OTP Computation

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

5.4.2.1. `computeOTPWithServerData`

Perform mobile side OTP computation.

```
NSError* error;
NSString* otp = [token computeOTPWithServerData:proxyData
message:message
credentials:credentials];
```

```
otpLength:length
error:&error];
```

Parameters:

- message - Message to compute the OTP.
- credentials - Credentials.
- otpLength - requested OTP length, 4-8 digits.
- error - Error reference.

5.4.3. Methods of the Refresh Session

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

5.4.3.1. createRefreshTokenRequestWithContext

Create a refresh request.

```
NSError* error;
NSData* context;
NSDictionary* refreshRequest = [token createRefreshTokenRequestWithContext:&context
error:&error];
```

Parameters:

- context – Operation context reference.
- error - Error reference.

5.4.3.2. updateRefreshTokenWithResponse

Handle the server response.

```
NSDictionary* refreshAdditionalRequest;
[token updateRefreshTokenWithResponse:refreshResponse
additionalRequest:&refreshAdditionalRequest
context:&context
error:&error];
```

Parameters:

- additionalRequest - (optional) Reference to operation additional request.
- context - Operation context reference.
- error - Error reference.

5.4.3.3. finalizeRefreshWithContext

Handle the server response.

```
BOOL b = [token finalizeRefreshWithContext:context
error:&error];
```

Parameters:

- error - Error reference.

5.4.4. Sample Code

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

```
\samples\OfflineOTP
```


6. Appendix

6.1. One-Time Password (OTP)

6.1.1. Sample Code using Direct Access

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).

6.1.1.1. Create the OTP Token

The type of the OTP is specified as follows:

Type of OTP	Value of the <code>andParameters</code>
HOTP	nil
TOTP	[[NSDictionary alloc] initWithObjectsAndKeys: @"totp", @"type", nil]

```

[[DYMobile sharedDYMobile]
createOTPTokenForUsername:@"username"
withCredentials:[[DYPINCredentials alloc] initWithPIN:@"1234"]
andParameters:nil // HOTP
completionHandler:^(DYToken *token, NSError* error) {
if (error!=nil) {
// Code deleted
}
}
}];

```

6.1.1.2. Compute the OTP Token

```
[[DYMobile sharedDYMobile]
computeOTPWithCredentials:[[DYPINCredentials alloc] initWithPIN:@"1234"]
completionHandler:^(NSString *otp, NSError* error) {
if (error!=nil) {
// Code deleted
}
}
}];
```

6.2. Controlling Global Properties

6.2.1. Controlling Log Level

By default, the level of the log is set to the "Warning".

Use the following to change log level:

```
[[DYMobile sharedDYMobile] setLogLevel:level];
```

The level parameter takes one of the following values:

- DYLogLevelDebug - all log messages
- DYLogLevelInfo - info, warning and error messages
- DYLogLevelWarning - warning and error messages
- DYLogLevelError - only error messages

Note

Logs are accessed using standard NSLog methods

6.2.2. Controlling Timeouts

The default Operation Timeout is 60 seconds. It can be reprogrammed by setting the following variable:

```
[[DYMobile sharedDYMobile].operationTimeout = N // seconds
```

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.

Value	Description
DY_EINVAL	Not valid argument.
DY_ENOENT	Not found error. Possible cause: <ul style="list-style-type: none"> Trying to use a token before creating one. Certificate file not found. SDK failed to communicate with the iOS Watch.
DY_ECOMM	Communication error. Possible causes: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> The device does not support fingerprint authentication. Error while checking the device fingerprint status.
DY_EINCORRECT_PIN	Incorrect PIN Possible causes: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Fingerprint sensor was added/removed from the device. Lock screen was disabled, re-create token (security issue).
DY_ETIMEOUT	Operation timeout reached.

Value	Description
	<p>Possible causes:</p> <ul style="list-style-type: none"> • Communication problem (server is not reachable). • The requested operation is taking more time than expected.
DY_ELOCKED	<p>Communication with the server is blocked by the protocol.</p> <p>Possible cause:</p> <p>Too many incorrect retries (wrong PIN).</p> <p>According to the server policy, there are two options:</p> <ul style="list-style-type: none"> • 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	iOS version is not valid.
DY_ETOUCHID_PASSCODE_NOT_SET	<p>Screen lock is not activated on the device.</p> <p>Possible cause:</p> <p>Using <code>DYFingerprintCredentials</code> on a device without enabling the screen lock.</p> <p>Suggestion:</p> <p>Enable the screen lock on the device and add at least one fingerprint before creating the token.</p>
DY_ERE_REFRESH_REQUIRED	<p>Mobile device exceeded the out-of-sync threshold for the token domain it is using.</p> <p>Possible cause:</p> <p>A deficiency in the connection to the server or in the finalization of the server response processing on the mobile device.</p>