TRACE4EU

KNOW YOYR CUSTOMER (KYC) TOOL

USE CASE

SOFTWARE SPECIFICATIONS

GOLDMAN SOLUTIONS & SERVICES

version: 3
10/11/2024
Nicosia

Contents

Overview

The problem

Customers are required to share their KYC (Know Your Customer) documents, either as a first time or as an updated version, several times and to several banks or several institutions that are doing business with. Each bank is forced to repeat the verification process irrespectively of whether this process has already been perfromed by other banks.

Objective

The objective of this use case is to allow a customer to "**upload once use many times**" of his KYC documents. It also speeds up the KYC verification process on part of the banks or other institutions, since a bank will be able to rely on the verification results already performed by other banks on a customer's KYC docs.

It **does not** provide a new process for KYC verification. This process remains the same as currently carried out by each bank or other institution.

Please check Future Enhancements section for suggested future functionality and improvements.

Technical overview

Banks wishing to take part in the KYC domain, make a request to CBC (Central Bank of Cyprus) to be onboarded to EBSI and become authorized to create new documents in TnT service on behalf of their customers.

Customers and banks **encrypt** private data, stored in off-chain or on-chain, using **randomly** generated **AES 256 bits** encryption keys. These encryption keys are kept encrypted in wallets local storages created and used by customers and banks.

Encryption keys are shared via TnT events and after being encrypted using the public key of the recipient. More precisely, the encryption keys are encrypted using an AES 256 bits encryption key **derived** from the ES256 private key of the sender and the ES256 public key of the recipient.

Customers, at any time, can delete their encrypted KYC docs they uploaded to off-chain storage.

User Flow

Below is the description of the steps and a flow diagram (see next page) of our suggested solution which is based on EBSI's TnT service.

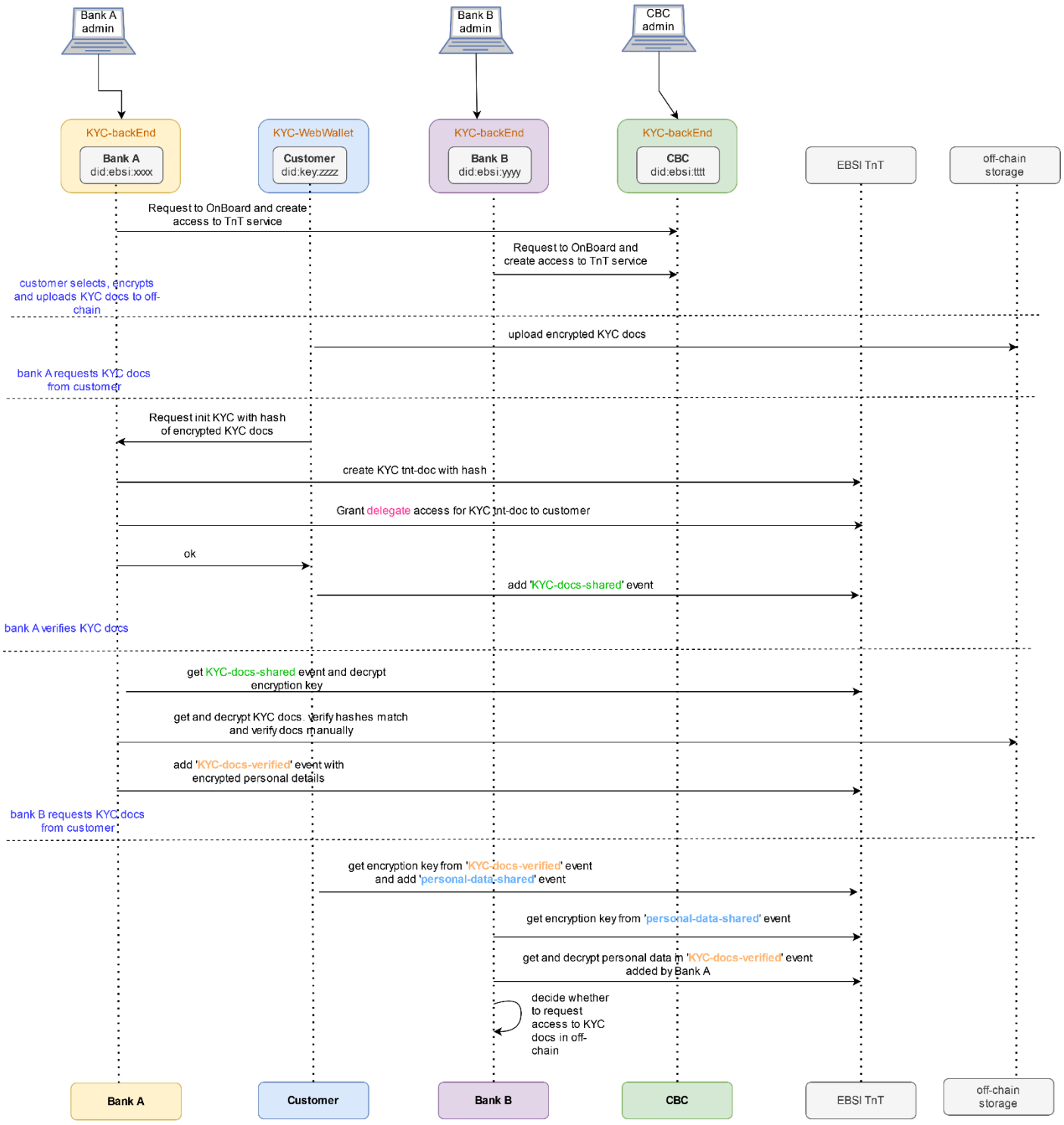
1. Banks wishing to take part in the KYC domain, make a request to CBC (Central Bank of Cyprus) to be onboarded and be allowed to create new documents in TnT service.

2. Customer A creates a web wallet, if doesn't have one, and chooses the wallet's option to upload his KYC docs to off-chain storage. Wallet pop-ups a file slecection window and lets the user select the docs he wishes to upload from his local drive. Once selected, wallet **encrypts the docs with a random encryption key**, and uploads them to off-chain storage.

3. Bank A makes an out-of-band request to customer A to share his KYC docs. Customer A chooses the wallet's option to share his KYC docs to Bank A. Wallet pop-ups a list of docs uploaded to off-chain storage and, after selecting one, wallet creates a hash of the encrypted docs and makes a request to Bank A's API to initiate the KYC docs sharing process passing the hash to the API.
4. Bank A's API makes a request to TnT to create a **new KYC tnt-document** for customer A using the encrypted docs hash.
5. Bank A grants delegate and write access for the newly created TnT document to customer A's **did:key** and returns a successful response to wallet.
6. Wallet updates the TnT document adding a KYC-docs-shared event for Bank A. The KYC-docs-shared event contains the KYC docs off-chain location url and the **customer's random encryption key** encrypted with bank's A public key.

7. Bank A's employee uses a web gui interface to get the encryption key from the KYC-docs-shared event and decrypt the customer's docs from the off-chain storage. He verifies that the hash of decrypted docs matches the TnT-id, manually validates the docs as of today and sends a request to TnT to update the document (writes a new event) with the verified customer's details (name, id, address, etc) **encrypted using a random encryption key.** The encryption key is also added to the event **encrypted using the customer's public key** (KYC-docs-verified event).

8. Bank B makes an out-of-band request to customer A to share his KYC docs.
   Customer A uses his wallet to update the TnT document (writes a new event) with the encryption key used by Bank A to encrypt his verified personal data **encrypted with Bank B's public key** (Personal-data-shared event).
9. Bank B's employee uses a web gui interface to get the encryption key from the TnT Personal-data-shared event and decrypt the customer's verified details included in the event KYC-docs-verified added by Bank A.
   Bank B's employee may decide to trust Bank A's verified details or ask the customer to grant him access to the original uploaded to off-chain KYC docs. In the second case, the customer will need to add a KYC-docs-shared event for bank B and grant write access to bank B in case bank B needs to add its own KYC-docs-verified event.


Bank A admin
Bank B admin
CBC admin
KYC-backEnd
Bank A
did:ebsi:xxxx
KYC-WebWallet
Customer
did:key:zzzz
KYC-backEnd
Bank B
did:ebsi:yyyy
KYC-backEnd
CBC
did:ebsi:tttt
EBSI TnT
off-chain storage
Request to OnBoard and create access to TnT service
Request to OnBoard and create access to TnT service
customer selects, encrypts and uploads KYC docs to off-chain
upload encrypted KYC docs
bank A requests KYC docs from customer
Request init KYC with hash of encrypted KYC docs
create KYC tnt-doc with hash
Grant delegate access for KYC tnt-doc to customer
ok
add 'KYC-docs-shared' event
bank A verifies KYC docs
get KYC-docs-shared event and decrypt encryption key
get and decrypt KYC docs. Verify hashes match and verify docs manually
add 'KYC-docs-verified' event with encrypted personal details
bank B requests KYC docs from customer
get encryption key from 'KYC-docs-verified' event and add 'personal-data-shared' event
get encryption key from 'personal-data-shared' event
get and decrypt personal data in 'KYC-docs-verified' event added by Bank A
decide whether to request access to KYC docs in off-chain
Bank A
Customer
Bank B
CBC
EBSI TnT
off-chain storage

Schemas

KYC TnT document

created by a bank following a request by customer to initiate KYC sharing.
in the request customer specifies his did:key and a hash of his **encrypted** KYC docs
Following tnt-doc creation, bank gives **delegate** and **write** permission to did:key

{
from: bank's ethereum address
documentHash: hash of encrypted KYC docs
documentMedata: "KYC doc created by Bank A"
didEbsiCreator: bank's did:ebsi
}

Grant delegate permission to customer's did:key

{
From: banks's Ethereum address
documentHash: hash of encrypted KYC docs = document id
grantedByAccount: UTF-8 of bank's did:ebsi
subjectAccount: secp256kl public key of did:key
grantedByAccType: **0** for did:ebsi
subjectAccType: **1** for did:key
permission: **0** for delegate
}

KYC TnT events

KYC-docs-shared event –
added by customer for a bank that requested access to his encrypted data in off-chain
{
from: customer's ethereum address
documentHash: document-id created by bank
externalHash: hash of string "KYC docs shared to Bank X"
sender: secp256kl public key of customer's did:key
metadata: KYC-docs-shared-metadata
}

KYC-docs-shared-metadata
{
eventType: 'KYC-docs-shared'
es256Did: wallet's es256 did. Need this for decryption
sharedForName: 'bank X'
sharedForDID: bank X DID
offchainFilepath: url of encrypted docs in off-chain
encryptedEncryptionKey: **customer's random secret key** encrypted with bank's X public key
}

Note: if bank X has no access to the TnT doc (it is not the creator of the TnT doc), customer must also provide write access to the bank so that bank X can also add a KYC-docs-verified event.

KYC-docs-verified event –
added by bank X
{
 from: banks's ethereum address
 documentHash: document-id created by bank
 externalHash: hash of string "KYC docs verified by Bank X"
 sender: UTF-8 of bank's X did:ebsi
 metadata: docs-verified-metadata
}

docs-verified-metadata
{
 eventType: "KYC-docs-verified"
 verifiedBy: "Bank X"
 encryptedEncryptionKey: **bank's X random secret key** encrypted with customer's public key
 //personalData **encrypted** using bank X's random encryption key
 encryptedPersonalData: {
  firstName:
  LastName:
  Id:
  Address:
  Salary:
  …
  …
 }
}

Personal-data-shared event –
added by customer for a 'KYC-docs-verified' by Bank X event for bank Y to see.
{
 from: customer's ethereum address
 documentHash: document-id created by bank
 externalHash: hash of string "personal data shared to bank Y"
 sender: secp256kl public key of customer's did:key
 metadata: personal-data-shared-metadata
}

personal-data-shared-metadata
{
 event-type: 'personal-data-shared'
 es256Did: wallet's es256 did. Need this for decryption
 verifiedBy: "bank X"
 sharedForName: "bank Y"
 sharedForDID: bankY DID
 docsVerifedEventId: eventid of KYC_docs_verified event added by bank X,
 encryptedEncryptionKey: **bank's X random secret key** encrypted with bank's Y public key

}

KYC BackEnd APIs

The KYC Back-end module is available for installation by both the Banks and the CBC.
A customizable environment variable determines its operation mode, BANK or CBC.

If operated by the CBC, banks can use CBC's APIs to request onBoarding to EBSI and granted write access to TnT EBSI service.

If operated by a Bank, customers can use the bank's APIs to share their KYC docs to the bank and the bank employee to verify the customer's docs.

Publicly exposed APIs

GET /getOnBoard?walletDID=bankDID

Offered by CBC module.

Called by a bank's backend module (bank's ent wallet) to initiate the issuance of **VarifiableAuthorisationToOnboard** vc from CBC. A pre-authorized credential offer will be returned.
The bank admin must present the pin provided by the CBC admin.
It is assumed that CBC's DID is already registered in TIR as a Trusted Issuer.
Banks can use this vc to register their DID in EBSI DIR.

POST /token

Urlencoded request:
{
grant_type: 'urn:ietf:params:oauth:grant-type:pre-authorized_code'
pre-authorized_code: pre-authorized_code from get_license_vc
user_pin: the provided **pin**
}

Finds an entry in banks db with pin and walletdid (in pre-authorised_code)
Generates access_token

Updates entry in db with access_token

returns
{
 access_token:
 token_type:
 id_token:
 c_nonce:
}

POST /credential

Bearer: **access_token** from above

raw request:
{
types: ["VerifiableCredential",
            "VerifiableAttestation",
            "VerifiableAuthorisationToOnboard"],
format: ‘jwt_vc’,
 proof : {
    proof_type: ‘jwt’,
    jwt: c_nonce signed by wallet
  }
}

Validates the submitted proof and access_token, checks existence of access_token in banks db and issues a VerifiableAuthorisationToOnboard vc

returns
{
  format: ‘jwt_vc’,
  credential: jwtvc
}

GET /getTnTaccess?walletDID=bankDID&bankName=name&bankUrl=url

Offered by CBC module.

Note: The CBC’s etherium address must be in **TnT allowed list** to be able to authorize others for TnT create access (EBSI support has to do this):

Called by a bank’s backend module (bank’s ent wallet) to request TnT create access. The bank’s onBoarding process must have been completed first.

CBC checks if DID is already authorized for TnT create. If yes, it skips the next step.

CBC checks if the bank's DID is in banks db and that it is registered in DIR and then authorises DID for TnT create.
==> run issueVcOnboard **CBCdid**
<self-issued-vcToOnboard>
==> resAuthTNT: authorisation auth tnt_authorise_presentation ES256 <self-issued-vcToOnboard>
==> using token resAuthTNT.access_token
 ==>tnt authoriseDid **CBC_DID bank_DID** true

Lastly, Updates the banks db with the bank's name and bank's backend module Url using bankDID

Note: EBSI support refused to white list the CBC's DID. This means that proving authorization for TnT create access to the banks will have to be performed manually, by opening a request to EBSI support.

Get /banks

Offered by CBC module.

Called by customer wallets to get a list of available banks to share their KYC docs to.
Returns a list from **banks** db of allowed banks to create KYC TnT docs with
bankName, bankDID, bankUrl.
The bankUrl can be used by customer wallets to connect to the bank's backend module.

POST /init_KYC_share

Offered by Bank module

Called by a KYC wallet to initiate his KYC docs sharing. In the request the wallet must include the hash of the encrypted KYC docs, its did:key and a self-signed empty vp_token to prove ownership of the did.

Req:
{
  documentHash: encrypted docs hash
  did: customer's ES256k wallet did:key
  customerName:
  vp_token: a self-signed ES256k jwtvp to prove ownership of did:key
}

A new TnT document will be created with the provided hash and then grant delegate and write access to the **es256k did:key** of the requesting customer's wallet.

It will also add in **tntInfo'** db the customer's name, wallet's did and documentHash.

POST /add_event

Offered by bank module
Called by KYC wallet to inform the bank that a new event has been added. **Not related to TnT events**.

Req:
{
 tntId:
 eventid:
 event_type: 'KYC_docs_shared'/'Personal_data_shared'
 customerName:
}

Stores tntId, eventid, event_type, customerName, status=pending in **events'** DB

Admin portal APIs

All admin's portal APIs require an access token as a bearer, issued from /login API. Called from the admin portal web GUI app.

POST /login

called to get an access token. Admin's Userid and password must be provided.

POST /genPIN

Offered by CBC's module
Generates a pin that can be used by a bank's employee to initiate the onBoard process for his bank.

Req:
{
bankDID:
}

Stores in banks' DB the bankDID and pin. An existing record with the same bankDID is deleted first, if exists.

Returns a pin which must be given to the bank employee to be used when calling the GET /getOnBoard API.

GET /walletcap

Offered by bank and CBC module

Displays the wallet's capabilities. DID, DID registry contents, TIR registry contents and if has TnT **create** access.

GET /newwallet

Offered by bank and CBC module

Generates a new ent did and private/public key pair for a new wallet setup
Following this, the admin must update the module's env file with the generated data.

GET /reqOnBoard?CBCurl=url&pin=pin

Offered by bank's module

Used by Bank's admin to initiate the onboarding process for the bank.

It calls the GET /getOnboard API on the CBC module url which will return a pre-authorized credential offer for an **VerifiableAuthorisationToOnboard** vc. It will then have to provide a pin to get the vc from CBC and then use it to register its DID in DIR registry.

Following a successful registration in DIR the GET /getTnTaccess API on CBC module will be called.

Note that for accessing TnT services is only required that a DID is registered in DIR. No need to be registered in TIR (ie no need to become a Trusted Issuer).

Detailed flow:

Bank bckEnd **CBC bckEnd**
/getOnBoard?walletDID ------ >
< ---------- credential-offer with

pre-authorized_code //expires in 24hours

/.well-known/openid-credential-isser --------- >
/.well-known/openid-configuration ---------- >
/token with
pre-authorized_code and PIN ------------ >
< ---------- access_token //expires in 24h. added in db rec.
//ok if record with pin && DID in
// pre-authorization_code
// exist in db
/credential with access_token and jwt proof ------ >
< ---------- vcjwt //uses access_token to get vc from db
//issues VerifiableAuthorisationToOnboard

Get accessToken from Autorization API using vcjwt
Register DID Document
Call GET /getTnTaccess on CBC

POST /decrypt_docs

Offered by bank's module

req: {

documentId:
eventid:
}

Used to get and decrypt the encrypted data on the off-chain storage associated with the specified eventId.

Get eventId
If event metadata = {event_type == 'KYC-docs-shared' && sharedForDID==myDID} then get encryptedEncryptionKey and offchainFilepath

Check if eventid.sender == did:key that has delegate access to TnTid (==customer's did:key)

KYC-docs-shared event contains **customer's random encryption key** encrypted with my public key in **encryptedEncryptionKey** property of its metadata.

encrypted_data = GET off_chain/download?file=url

calculate hash of encrypted_data and verify it is the same with tntId

customer_key = decrypt encryptedEncryptionKey with my private key and wallet public key

clear_data = decrypt encrypted_data with customer_key

return clear_data

POST /decrypt_personal_data

Offered by bank's module

req: {

documentId:
eventid:
}

Used to get and decrypt the encrypted personal data in a personal_data_shared event.

Get eventId
If event metadata = {event_type == 'personal_data_shared' && sharedForDID==myDID} then get encryptedEncryptionKey and docsVerifedEventId.

Check if eventId.sender == did:key that has delegate access to TnTid (==customer's did:key)

personal_data_shared event contains **somebank's random encryption key** encrypted with my public key in **encryptedEncryptionKey** property of its metadata. Also, In **docsVerifedEventId** contains a reference to KYC_docs_verified event which contains the encrypted personal data added by somebank.

Get docsVerifiedEventId

encrypted_personal_data = KYC_docs_verified.EncryptedPersonalData

somebank_key = decrypt **encryptedEncryptionKey** with my private key and wallet public key

clear_personal_data = decrypt encrypted_personal_data with somebank_key

return clear_personal_data

POST /mock_decrypt_docs

Offered by bank's module

req: {
offChainFile:
encEncKey: encrypted encryption key
walletDID: es256 did
}

Used to **test** decryption of off-chain encrypted docs bypassing TnT access.

encEncKey contains **customer's random encryption key** encrypted with my public key

encrypted_data = GET off_chain/download?file=url

customer_key = decrypt encEncKey with my private key and wallet public key

clear_data = decrypt encrypted_data with customer_key

return clear_data

POST /kyc_verified

Offered by bank module

Bank admin adds a KYC_docs_verified event to a TnT doc following a successful manual verification of some KYC docs in off-chain storage.

Req:
{
tntId:
personalData: {
firstName:
LastName:
Id:
Address:
Salary:
...
...
}

}

Get TnT doc and check if I am the creator or have been given write access by the customer.
Encrypt personalData with a random key. personalData is the result of the manual verification process.
EncryptedEncryptionKey: the **random key** encrypted with customer's public key
EncryptedPersonalData: personalData encrypted with **random key**

event_data = {
event_type: 'KYC_docs_verified',
verifiedBy: 'my Name',
EncryptedEncryptionKey,
EncryptedPersonalData
}

Call add_event_inTnT(tntId, event_data)
Stores tntId, eventid, event_type= KYC_docs_verified , customerName, **randomkey**, status=completed in events' DB

GET /kyc_tnt_docs?bankName=name

Offered by bank's and CBC module
Returns a list of all kyc tnt docs created by this or any other bank

Get all TnT docs where documentMetadata includes "KYC"

If bankName exists filter results where documentMedata includes bankName

If bankName == myName

  For each entry in resulted list add customerName from **tntInfo** db

Return list with tntId, createdBy, customerName (optional)

GET /kyc_tnt_doc/:id

Offered by bank's and CBC module

returns a json of the tnt doc details and of all its events details

GET /events?status=completed/pending/all

Offered by bank's module
Returns a list of events from **events** db.

Not related to TnT events.

KYC front-end (Admin portal)

A reactJs web application accessed by the bank or CBC admins.

On loading displays login page. User enters userID/password and calls the POST /login API on KYC backend. The /login API returns an access token which is saved in session storage.
On successful login go to Get Events page, if a bank operated module, and Get KYC docs, if a CBC module.

Menu Options displayed on a vertical menu bar on left of the page:

Get Events (for bank module only)
Get KYC docs
Req OnBoard (for bank module only)
Gen Pin (for CBC module only)
View Wallet
New Wallet
Logout

Get Events

Available to a bank admin

Title “check events submitted by KYC customers”

Gives the option to select **one** of the following:

- Pending events
- Completed events
- All events

When the ‘**Get Events’** button is pressed:

Call GET /events?status=pending/completed/all from **local events** db.
Display a list with event-type, customerName, and status (if all)
The list also includes tntId and eventid but not displayed.

User can select one from the list. On selection:

if KYC_docs_shared event then give option to

a) Get off-chain docs
b) add KYC_docs_verified event – only if status == pending
c) Mark as completed – only if status == pending

If Personal_data_shared event then give option to

a) View verified personal data
b) Request off-chain data access -> sends email to customer
c) Mark as completed – only if status == pending

if KYC_docs_verified event (added by myBank) then give option to

a) view verified data

Get KYC docs

Available to bank and CBC admins

Title “check for KYC TnT docs created by me or any other bank”

Call get /banks CBC API

Created by : drop down list of available banks + ‘all’

Call get /kyc_tnt_docs?bankName=selection //if all is selecte no parameter is added to the query

Display list with createdBy, date, customer Name (only if bankName==myName)

User can select one from the list and press ‘**show’** button

When show is pressed call get /kyc_tnt_doc/tntId

KYC Wallet

A standalone reactJs web application created and used by customers.

On loading, it checks the browser's local storage for the existence of wallet keys. If not found it offers the option to "Create a new Wallet". If found, it displays the login page.

If create new Wallet is selected, it asks to enter/confirm a password and a new DID (did:key:xxx) and a pair of **ES256** public/private keys is created and stored encrypted in local browser's storage. A second DID and key pair is also generated for **ES256K**. Both key pairs are generated from the same random private HEX key. The user's password is used to encrypt data stored in local storage.

Login Page

It displays an entry field for the password.

There is no server login. **Only local login**. When a password is selected during wallet creation the word "success!" is encrypted with the password and stored in local storage with the key= "access". During normal login the password entered is used to decrypt the phrase stored with the key "access". The login is successful if the decrypted word is "success!". If successful it sets the isLoggedIn variable to true.

Following a successful login, it displays the following options on a vertical side menu bar:

My Wallet
My Name
My Activity
Prepare KYC docs
Upload KYC docs
Share KYC docs
Share my Vertified data
Logout

On top left of the page, it also displays the customer's name from myname key in local storage.
If the key does not exist in local storage it displays '*please add your name using My Name option*' in red.

On the page's body the My Activity screen is displayed. see option below.

My Wallet

Displays ES256 and ES256K DIDs and key pairs from local storage. No mnemonic phrase is used.

My Name

Title "Add or Change my name"

Displays the value in key=myname in local storage, if exists
If it doesn't exists it shows an empty entry field with a '**add name**' button
If it exists it displays the existing name in an entry field with a '**change**'button.
If added or changed the display on left top of the page is updated.

My Activity

Title "Activities performed with my KYC docs"

User is asked to select one of the following options:

- Docs uploaded to off-chain
- Docs shared to banks
- My verified data by banks

when **'proceed'** button is selected:

Docs uploaded to off-chain option:

Gets and decrypts key=off-chain array from local storage
Displays a list with file-names. User can select one from the list.
When an entry is selected and '**show'** button is pressed the offchain encrypted data is fetched using the offchainFilepath and decrypted using the random key in local storage.
The decrypted doc is displayed
When an entry is selected and '**delete'** button is pressed the off-chain doc is deleted from off-chain storage and from local storage array.

Docs shared to banks option:

Gets and decrypts key=off-chain array from local storage
For each Hash in the array
  Get TnT doc
  If doc exists and event-type == "KYC_docs_shared" event exists
    Add to list file-name (in array), shareForName, offchainFilepath (in event)

Display the created list (omit offchainFilepath). User can select one
When an entry is selected and '**show'** button is pressed the offchain encrypted data is fetched using the offchainFilepath and decrypted using the random key in local storage.
The decrypted doc is displayed

My verified data by banks option:

Gets and decrypts key=off-chain array from local storage

For each Hash in the array
Get TnT doc
If doc exists and event-type == "KYC_docs_verified" event exists
Add to list file-name (in array), verifiedBy (in event), eventId

Display the created list (omit eventId). User can select one
When an entry is selected and '**show'** button is pressed
Get eventId
Decrypt EncryptedEncryptionKey and and get bank's random encryption key
Decrypt personal data with bank's random encryption key
show personalData

Prepare KYC docs

Title "Prepare KYC docs for upload"

When '**select files'** is chosen user is asked to select one or more files (pdf or image) from local drive.
When '**prepare'** is chosen, selected docs are combined to one file called
KYC_docs_ddmmyyyyy_hhttss.pdf and saved in local drive

Upload KYC docs

Title "Encrypt and upload my KYC docs to off-chain storage"

When '**select Prepared File'** is chosen, user is asked to select one of the already prepared KYC doc.
Only files which start with "KYC_docs_*" are shown for selection.
When '**upload'** is chosen a **random AES key** is generated and the selected file is encrypted with this key.
The encrypted file is uploaded to off-chain storage by calling Post off_chain_url/upload with a self-signed vp_token. The offchainFilepath is returned.

Hash of encrypted file is calculated.
prepared_file_name, random-enc-key, hash, and offchainFilepath is added in local storage array with key=**off_chain**

Share my KYC docs

Title "Share my KYC docs on off-chain storage to a bank"

Calls get /banks CBC API
The entries fields are displayed:

Bank to share to : drop down list of available bank names
KYC docs to share: drop down list of file names from local storage with key=off-chain

When '**Proceed'** is pressed
Get selected bank's API endpoints
Check if TnT doc with hash of selected file already exists
If not, Call POST /init_KYC_share with myName, hash of encrypted file, vp_token – **bank creates TnT doc and grants delegate and write access to my es256k did:key**

If yes, check if bank has write access to tnt. If not, grant write access to the bank

encryptionKey = get random-key from local storage associated with selected file.
call GET /jwks on the selected bank's url to get bank's public key
encryptedEncryptionKey = random-key encrypted with selected bank's public key.
call add_event_inTnT('KYC_docs_shared', tntId, encryptedEncryptionKey)

Call POST /add_event on selected bank's url //this is only added to bank's events db not to TnT doc

Share my Verified data

Title "Share my verified personal data to another bank"

For each hash in local storage off-chain array
Get TnT with tntid==hash
If event_type == 'KYC_docs_verified" exists add file-name, tntId, eventId, verifiedBy to list
where file-name is found in local storage with hash == tntId
if list is empty give pop-up window message "*you have no verified KYC docs*" otherwise proceed with the below.

Calls get /banks CBC API

The entries fields are displayed:

Bank to share to : drop down list of available bank names – exclude verifiedBy bank
Verified data to share: drop down list from above list with file-name, verifiedBy

When "**show verified data**" button is selected
Get eventid
Decrypt encryptedEncryptionKey in KYC_docs_verified and and get bank's random encryption key
Decrypt personal data with bank's random encryption key
show personalData

When '**Share personal data'** is pressed
Get eventid
Decrypt encryptedEncryptionKey in KYC_docs_verified and and get bank's random encryption key.
encryptedEncryptionKey = bank's random encryption key encrypted with selected bank's public key
Call add_event_inTnT('Personal_data_shared', tntId, encryptedEncryptionKey, eventid of KYC_docs_verified)
Get selected bank's API endpoints
Call POST /add_event //this is only added to bank's db not to TnT doc

Off-chain storage

A simple cloud server that provides upload and download file APIs.

Customers can use the upload API to upload their **encrypted** KYC docs. In addition to the file contents, a signed empty **vp_token** must be provided. The server will associate the stored file's path with the did:key of the sender, following validation of the key:did ownership.

Anybody can download a file but without the random AES key that was used to encrypt the file, the file can not be viewed.

A customer can also delete a file that was previously uploaded. In order to do this, a signed empty vp_token needs to be provided to the server by the wallet requesting deletion. The server will verify ownership of the did:key and that the file for deletion is associated with the requesting did:key and delete the file.

Future Enchancements

New customers -open account

A bank, having verified the KYC docs for a new customer, will be able to allow the new customer to open an account with the bank and create an account for him on its portal.

The customer, using his EBSI KYC wallet, will present to the bank a self-signed vp_token and the TnT id associated with his **KYC_data_verified** event. After verifying that the did:key belongs to the requesting wallet, and the did:key has delegate access to the submitted TnT id, the bank backend module will provide an access token to the wallet which will use to call an API for opening an account for the customer.

New customers – VCID presentation required

A bank may require to associate the submitted KYC docs with the user's identity from a trusted authorisation service before it can proceed with the actual verification of the KYC docs. In this case, the init_KYC_share API will require the submission of a **VCID vp** that the customer can request and get issued from the government of Cyprus using his KYC wallet.

Existing customers – KYC docs update

If an existing customer is asked to update/re-affirm his KYC documents as part of CBC regulations, a bank may require the customer to **first login** to its portal with his existing credentials before it allows him to call the init_KYC_share API.