unbound security

Unbound CORE Crypto Asset Security

Developers Guide

Version 1.0.2106

October 2021

Table of Contents

Copyright 2021, Unbound Security

Copyright 2021, Unbound Security

Copyright 2021, Unbound Security

1. Revision History

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

Version	Date	Description
1.0.2106	October 2021	Minor updates from rebranding.
1.0.2103	May 2021	Added section on Public Key Encoding.
1.0.2010	January 2021	Updated the API Reference with the new API sections. Added section CASP Java SDK containing details about developing participants and data collectors.
1.0.2007	September 2020	Updated Whitelisting with a new parameter.
1.0.2004	May 2020	Updated Audit Reports with a new script that resets the audit data.
1.0.2001	January 2020	Added section Built-in Wallets with Custom Chain Connectors. Updated the Authentication section. Added CASP Transaction Handler. Added information about Whitelisting. Added section Audit Reports.
1.0.1910	November 2019	Updated APIs for offline bot participants. Information about offline bots can be found in the Offline Bots section of the CASP User Guide.
1.0.1906	September 2019	Updated Participant Status with new statuses. Added section CASP Operators and Participants. Added section CASP Roles. APIs were updated for CASP 1.0.1906.
1.0.1905	July 2019	APIs updated for CASP 1.0.1905.
1.0.1904	June 2019	Initial version.

Copyright 2021, Unbound Security

2. CORE CASP Solution Overview

Unbound CORE Crypto Asset Security provides the advanced technology and the architecture to secure crypto asset transactions. The crypto asset solution contains the CASP service and different endpoints (humans or bots). This solution is referred to as the Unbound CORE Crypto Asset Security Platform, or **CASP**.

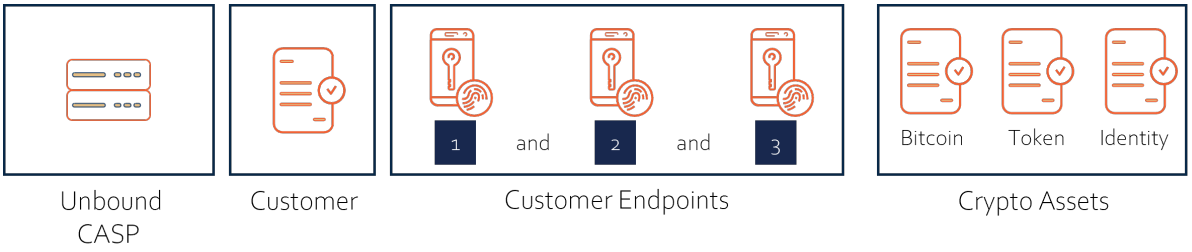


Figure 1: CORE Crypto Asset Solution Components

The CASP solution is built on the technological foundation of secure multiparty computation (MPC). As used in the CASP solution, it provides the following benefits:

- A Risk-Based Quorum Policy that provides a flexible mechanism to handle transaction signing by multiple participants across multiple groups.
- Each private key exists as several separate random shares stored on separate locations and is refreshed constantly.
- Key material never exists in the clear at any point of its lifecycle. Key shares are never combined at any point in time – not even when used or when created.
- An attacker needs to get control over all involved servers and clients, simultaneously.
- Real-time, tamper proof audit log that logs any key operation.

2.1. CASP Components

CASP provides the framework to create Accounts, which hold multiple Vaults where secrets are stored. Access control is managed by the Risk-Based Quorum Policy for all of the Participants.

2.1.1. Accounts

An account is a container for a set of vaults and participants that manage these vaults. An account may represent a customer of the system, a trader, an organization, etc. The CASP service supports creating different accounts, managing account participants (human users or machine bot's), creating secure vaults for the account, and executing different crypto asset transactions within the account.

The CASP service supports the notion of global accounts, which can manage vaults across other accounts. This notion may support the use case of the CASP service providers managing vaults on behalf of the CASP customers.

Copyright 2021, Unbound Security

2.1.2. Participants

A participant can be a human within the account or a bot taking part in crypto asset transactions. Each participant owns a share of the cryptographic material that is part of the different transactions. CASP supports participants using any relevant platform, including mobile devices, laptops, and different server platforms for bots. Participants can be hot or cold, where cold participants are not connected to the internet.

2.1.3. Vaults

A vault is a secure container for the cryptographic material used to protect a crypto asset, such as the seed or private key. CASP uses Multiparty Computation (MPC) to split the crypto material between the different participants in the vault, which ensures that the material never exists in a single place. In addition, only the approved set of participants can complete a transaction based on the vault definition.

A **Quorum** vault shares the responsibility of executing a transaction between many different participants in a structure defined by the vault policy. The vault policy contains a quorum-based structure where there are any number of groups, any threshold value per group, any tree structure between different groups, etc. The MPC protocols used by CASP ensure that if and only if the quorum definition is satisfied, a transaction can take place, which is enforced on the cryptographic level.

2.1.4. Risk-Based Quorum Policy

Each CASP vault has a set of risk-based quorum policies associated with it, which are defined during vault creation. These policies assign a different quorum policy to different transactions, based on the transaction details (such as the transaction amount or the time of day).

Approvals are defined by a group of authorizing entities, of which a minimal-size subset (called a quorum) is required to approve the transaction. M approvals from a set of N entities is known as "MofN". For example, the client may define 8 entities, of which 4 must approve the transaction. Another example is where there must be 3 approvals from group A and 2 approvals from group B.

The number of groups, size of groups, and the size of the approving subset is fully flexible and can be different for each vault.

2.1.5. Admin Quorum

A risk-based policy vault requires the definition of an admin quorum, i.e. a quorum of participants that approve any change to the policies of this vault. By defining such an admin quorum, CASP assures that any change to any policy is reviewed by MofN participants.

2.1.6. Data Collectors

Data collectors are independent components that calculate policy related **attribute templates** (custom static attributes) for transaction signing. Each data collector is associated with an **attribute template group** that contains the attribute templates.

Copyright 2021, Unbound Security

Unlike participants, which can be human and require no development, data collectors by definition require development by the customer.

2.1.7. Synchronous and Asynchronous Operations

CASP is a collaboration service, where different participants collaborate to perform crypto asset transactions. As such, it has inherent support for asynchronous operation. When an operation is triggered, it is located in a queue and completed when the relevant set of participants complete their part. CASP supports triggering asynchronous operations, notifying the relevant participants on required actions, and checking the status of operations.

2.1.8. Blockchain

CASP can be used for securing the crypto material and managing crypto transactions where communication with the different blockchains is external to the CASP system. In such a case, CASP is mainly used for securing the vault keys and executing the sign operation. CASP can also support managing the blockchain operations itself. In such a case, CASP is capable of showing balances, sending transactions to the blockchain, checking blockchain transaction status, etc.

2.2. Using CASP to Sign a Transaction

The general flow for CASP signing a transaction is as follows:

1. A user wants to make a transaction and triggers a request to the crypto asset system.
2. The crypto asset system exchange triggers a request to the CASP Orchestrator for approval.
3. The CASP Orchestrator communicates with all quorum members to obtain the required partial signatures.
4. The CASP Orchestrator then can write the signed transaction into the ledger or return it to the application that called it.

2.3. Solution Architecture

The CASP architecture is shown in the following figure.

Copyright 2021, Unbound Security

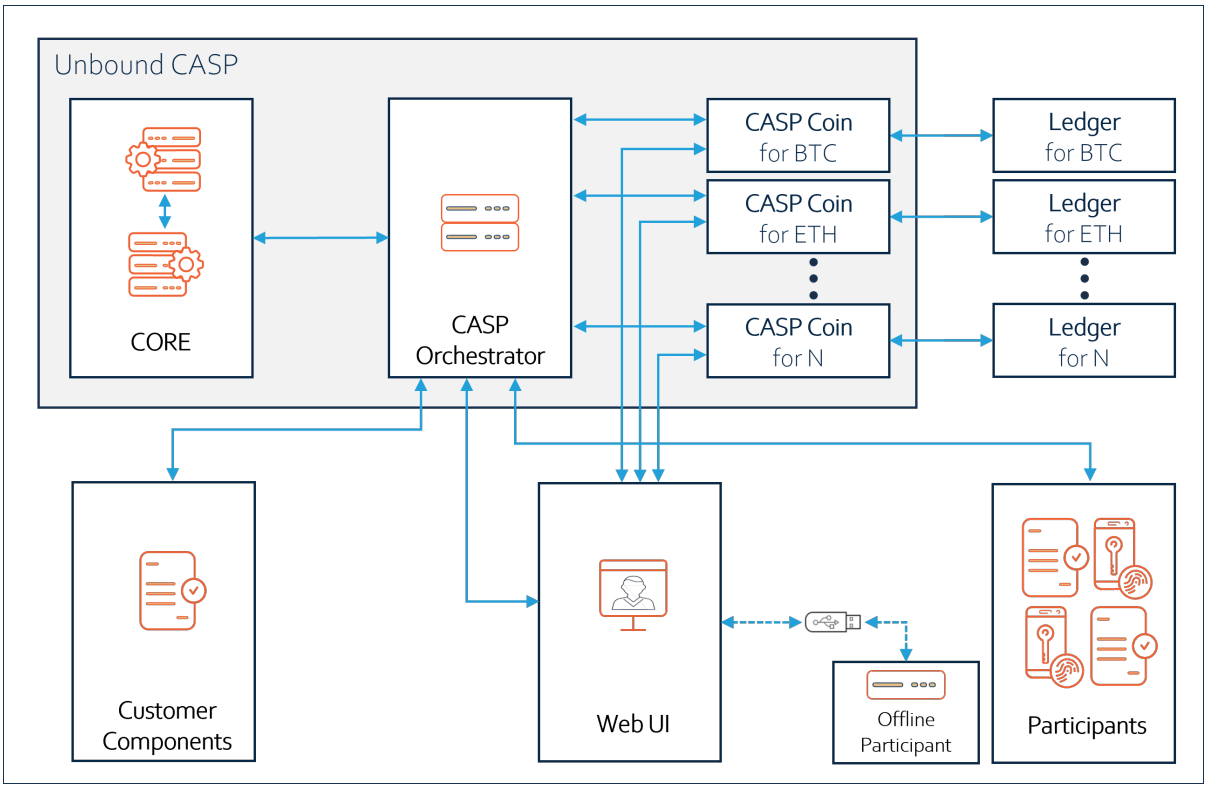


Figure 2: CASP Architecture

The **CASP Orchestrator** is the heart of the CASP system. It communicates with all parts of the system, initiates creation of the key shares for the vault, manages the different distributed procedures, and acts as the external entry point for the relevant applications, such as the crypto asset applications.

The CASP Orchestrator is backed by the powerful key management capabilities of Unbound CORE Information Security ("CORE"). CORE works together with the **Participants** to provide the complete approval signature for transactions, where Participants can be mobile devices, desktops, servers, or bots. CASP creates an ECDSA key, EdDSA key, or Schnorr key, which is used for all transactions, along with support for BIP derivations.

CASP's open architecture enables communication with different types of crypto asset ledgers. CASP uses the term *wallet* (also known as *chain adapter*) to refer to the component that communicates with the ledger. For example, the BTC wallet enables communication with a Bitcoin ledger. It prepares a transaction from the available ledger data and sends it to CASP. CASP then signs the transaction and returns it, which then transmits the signed transaction to the ledger. CASP seamlessly works with many different types of ledgers.

Copyright 2021, Unbound Security

3. Which API Do You Need?

To use the CASP API, follow these steps:

1. Select the implementation scenario of the CASP API that you need:
    - If you want to use the built-in CASP wallets (supporting Bitcoin, Ethereum, and ERC-20), use the **Built-in Wallets API**.
    - If you want to implement your own full custom wallet and use CASP for key protection and quorum signing use the **BYOW API**.

    > **Note**
    > If you are designing for multiple coin types, it is possible that you use multiple implementation scenarios.

    These implementation scenarios are explained in the following sections.
2. Read the General API Information.
3. Consult the detailed API Reference for your implementation scenario.

3.1. Built-in Wallets

CASP provides built-in support for different wallets. A wallet contains a specific crypto currency that has its own associated ledger where transactions are recorded. CASP supports the following wallets:

- btc - bitcoin.
- btctest - bitcoin Testnet3.
- eth - Ethereum.
- ethtest - Ethereum testnet, for networks supported by Infura.
- ERC-20 - Ethereum and Ethereum testnet blockchain tokens that are compliant with ERC-20. A list of supported tokens can be found here.

See the CASP User Guide for information on how to configure these wallets.

3.2. Built-in Wallets with Custom Chain Connectors

CASP provides plugins that can be adapted to different types of crypto asset ledgers.

3.2.1. Prerequisites

- Node-js 10.17.0
- typescript (run `npm i -g typescript`)
- ts-node (run `npm i -g ts-node`)
- Unbound Wallet GitHub repository - contact Unbound Support to get access.

Copyright 2021, Unbound Security

> **Note**
> The links in the following steps only work after you have access to the GitHub repository.

3.2.2. Setup and Sanity Test

1. Clone the repository:

```
git clone https://github.com/unboundsecurity/wallets
```

2. Navigate to the wallets directory that you downloaded.

```
cd wallets
```

3. Install the project.

```
npm i
```

4. Run the sanity tests. All should pass.

```
npm test
```

To add your own plugin, please follow these steps:

1. Create a folder under `src/blockchainProviders` with the new chain connector name. For example: `src/blockchainProviders/xrp`
2. Create a file implementing BlockchainProvider, such as `XrpProvider.js`. This file should export one class, which is the blockchainProvider that implements `_getLedgers()` and `getId()`.
3. Create an `index.js` file in the provider folder that references and exports your blockchain provider file.
4. Implement one or more instances of LedgerAdaptor and return them from `_getLedgers()` in your provider.
5. Edit the configuration file and add your provider under `chainProviders`.
6. Change the `useLedger` attribute in your configuration file to the specific ledger name that you want to use. For example: `XRP-TEST`. A ledger with that ID must be returned from `getLedgers` from your chain connector.

Copyright 2021, Unbound Security

7. Run the demo.
    a. Navigate to the *src* directory.

    ```
    cd src
    ```

    b. Create your configuration file by copying the template in `/config/cli/config.js` and adding configuration for your chain connector.

    c. Run:

    ```
    node ./cli.js --config=<path-to-your-config-file>
    ```

    If no *config* is specified, the default config file in */config/cli/config.js* is used.

    d. Follow the instructions on the screen. The demo is designed to require minimal input from the user.

Refer to the README in Github for more information.

3.3. Bring Your Own Wallet (BYOW)

CASP provides the necessary APIs so that you can bring your own wallet (BYOW), meaning that you can use whatever ledger you have, and control the vault and key operations with CASP. Using BYOW, you can create an implementation that can handle any coin type, as well as any special operations that you use to communicate with your ledger and for ledger processing.

You may want to use BYOW when using one of the built-in wallets, but need other features, such as using your own nodes and address caching. You may already have management in your application for one of the built-in wallets and want to use CASP for key management and protection. BYOW provides the flexibility to be used with any blockchain application.

In these scenarios CASP handles key management, protection and signing operations based on the quorum specification. It is your responsibility to manage:

1. Blockchain communication
2. Building transactions
3. Balance management and retrieval

Setup the basic system components, accounts and participants, using the CASP APIs. Once the relevant participants exist, you can start creating your own vaults.

Copyright 2021, Unbound Security

4. General API Information

Integrating CASP services within your server application, such as an exchange, a custodian service, or any other application managing crypto assets, is accomplished with the CASP REST API.

This reference includes operations for completing the full CASP life cycle, including:

1. Creating accounts
2. Adding participants to an account
3. Creating vaults
4. Depositing money into vaults
5. Withdrawing money from vaults
6. Managing keychains

See the Unbound CASP User Guide for details about installing CASP.

4.1. REST API Requests

The API prefix for all CASP REST endpoints is:

`https://<casp-server>/<comp>/api/v1.0/`

The value for `<comp>` is the name of the coin type, such as *btc*, or *casp*.

4.2. Error Handling

Responses are formatted in the standard REST format, with a *status* field showing an error code and an *error* field with a text description of the error. The possible error codes are described with each API.

For example, here is an *apikeys* request:

```
https://<casp-server>/casp/api/v1.0/mng/auth/apikeys
```

The error that is received is shown on the right.

```
Date: Mon, 05 Nov 2018 07:49:10 GMT
Content-Type: application/problem+json
Content-Length: 121
{
 "type":"/mng/errors/operation-failed",
 "title":"Authentication failed",
 "details":"Password must be changed",
 "status":400
}
```

4.3. CASP Operators and Participants

CASP has both operators and participants.

Copyright 2021, Unbound Security

1. **Operators** - manage users, participants, and vaults. They initiate operations and manage policies. Each operator has a role, as described in CASP Roles.
2. **Participants** - members of quorum groups that can approve or decline operations. See Participant Flows for the actions that participants are involved in.

4.4. CASP Roles

Each operator in CASP has an associated **role**. This role defines which actions are permissible for the operator to execute and what data the operator has access to.

Roles can be one of the following:

- **Security Officer** - has permissions associated with system management, such as:
  - User management - create new operator and reset operator passwords.
  - Account management - create and edit accounts.
  - Participant management - create and edit participants, generate activation codes.
  - Create backups.
- **Trader** - has permissions associated with vaults and transactions, such as:
  - Operations - start the signing process, and approve transactions.
  - Keychain management - create an address, derive a key.

    Note
    The Trader cannot create new operator or participants.
- **Super User** - has all permissions.

  Warning
  Use extra caution with the Super User role, since it has all available permissions.

4.5. Public Key Encoding

Some CASP operations, such as List chain addresses, provide a key encoding parameter that determines the key format. The public key encoding can be one of the following:

- DER - – ASN.1 encoded (for ECDSA or EdDSA)
- compressed – provides the x coordinates and the sign of the y coordinate (ECDSA only).
- uncompressed – provides the x,y coordinates of the point on the curve (ECDSA only).
- plain - provides the x,y coordinates of the point on the curve (EdDSA only).

4.6. Proof of Ownership

Proof of ownership can be achieved by verifying a signature returned by a CASP sign operation. This action proves to someone else that you hold the private key associated with the vault.

The proof of ownership process is as follows:

Copyright 2021, Unbound Security

1. Produce a hash for signature from any plaintext using bitcoin magicHash function.
2. Create a CASP sign request with the getSignOperation endpoint to sign the produced hash.
3. From the response data received from CASP, extract the signature and recovery code.
4. Send the signature, original plaintext message, recovery code, and address to the verification entity.
5. The verification entity can verify the address and signature using the recovery code.
6. The verification entity should run the bitcoin verify message algorithm.

For a sample of the verification process, see Proof of Ownership in GitHub.

4.7. Accounts

Click here to open the Account APIs.

An account is a container for a set of vaults and participants that manage these vaults. An account may represent a client of the system, a trader, an organization, etc. The CASP service supports creating different accounts, managing account participants (human users or machine BOT's), creating secure vaults for the account, and executing different crypto asset transactions within the account.

The CASP service supports the notion of **global accounts**, which can manage vaults across other accounts. This notion can support the use case of the CASP service providers managing vaults on behalf of the CASP clients.

4.8. Attributes

Attribute templates enable you to add custom static attributes to a CASP vault during vault creation. Attributes consist of one or more of the following types: string, numeric, date and Boolean. To ensure security, the attributes are approved by the admin group during vault creation. A customer can use these attributes to add logic to a policy. For example, a transaction that is initiated during a predefined date range will go through a certain MofN approval policy.

4.9. Audit Reports

Click here to open the Audit Report APIs.

Audit reports can be generated detailing account, user, vault, and sign operations. There is also the capability to delete old data.

These reports are tamper-proof, using a scheme where each line is signed. An endpoint is provided to verify the integrity of the audit report.

A script is also provided with the installation that resets the audit data. If needed, run the script with this command:

```
sudo casp_delete_ukc_secrets
```


Copyright 2021, Unbound Security

4.10. Authentication

Click here to open the Authentication APIs.

Most endpoints require authentication to submit a request to the CASP server. The API reference for each endpoint specifies the types of authentication needed to access the endpoint.

Endpoints in the Getting Started section do not need authentication. This allows you to easily retrieve the system status.

CASP uses username/password and API keys to add a layer of security to the API requests from users (see CASP Users and Participants for a definition of *users*).

4.10.1. Username and Password

For **short term** access, such as for the web UI, the user can use a username and password to generate an **access token** that is valid for 20 minutes by calling the Get access and refresh token endpoint. The access token must be added to the authorization header (as a bearer token) to every REST call.

To get or refresh the access token, call the Get access and refresh token endpoint. The following code samples show the types of authentication that can be used with this endpoint.

1. Using a username/password in an Authentication header.

```
curl --request POST \
 --url http://localhost:8888/casp/api/v1.0/mng/auth/token \
 --header 'authorization: Basic c286Y2FzcCBydWxleno=' \
 --header 'content-type: application/json' \
 --data '{
          "grant_type":"password"
 }'
```

2. Using username/password in the body.

```
curl --request POST \
 --url http://localhost:8888/casp/api/v1.0/mng/auth/token \
 --header 'content-type: application/json' \
 --data '{
          "grant_type":"password",
          "client_id":"so",
          "password":"password"
 }'
```

4.10.1.1. Refresh the Token

Once you have an access token, you can use the same endpoint to refresh the token.

This is an example of refreshing the token:

```
curl --request POST \
 --url http://localhost:8888/casp/api/v1.0/mng/auth/token \
 --header 'content-type: application/json' \
 --data '{
          "grant_type":"refresh_token",
```


Copyright 2021, Unbound Security

```
        "refreshtoken":"c386ZjU4NGE1ZjUtNjRlOS00M1M9LWIyOTItOWYzZjc0NjUxODg6"
}'
```

4.10.2. API Key

For applications or scripts that require **long term** access, an **API key** with a one-year time limit can be created. The CASP Admin first creates an access token and then uses it to generate an API key by calling the POST /auth/apikeys operation (and providing the credentials). The returned API key must be added to the authorization header (as a bearer token) for every REST call. The Admin can generate several API keys for different app usages.

4.10.3. Token Usage

Once you have an access token or API key, you can call endpoints using it, such as:

```
curl --request GET \
--url http://localhost:8888/casp/api/v1.0/mng/auth/users \
--header 'authorization: Bearer
dXNlcjExMTE6NWRmYWQwN2EtODgyMi00NzM0LTg4NTMtYjA4YjhkNTc0ZTYx'
```

The *header* field has the format:

```
--header 'authorization: Bearer <token/key>'
```

The <*token/key*> field can be the access token or the API key.

4.11. Backup

Click here to open the Backup APIs.

CASP supports backing up key material and restoring the backup information for the different levels of the CASP entities: per vault, per sub-account (for HD wallets), and per specific address (for HD wallets).

When backing up a key, the CASP backup mechanism supports a publicly verifiable backup. Using only the backup public key, you can verify that the encrypted data is indeed the EC private key, which matches the known EC public key.

Three APIs are provided for backups:

- Backup a single key in a deterministic (Non-hierarchical) vault: Get vault backup data
- Backup a sub-account key in an HD (BIP44) vault: Get backup for account
- Backup a specific key in an HD (BIP32) vault: Get public key backup data

4.12. General Status

Click here to open the General Status APIs.

General endpoints are provided to check the health of the CASP service, the supported ledgers, etc.

Copyright 2021, Unbound Security

4.13. Identity Providers

CASP can leverage CORE integration with the OpenID Connect (OIDC) providers enabling the Single Sign-On (SSO) authentication for the CASP partition users. The CASP UI login page presents the standard login option and a list of the registered SSO providers. To log in, specify the required partition, select the personal authentication option, and proceed as directed.

4.14. Keychain Management

To open the Keychain Management APIs, click for Built-in Wallets or BYOW.

Some implementations require a higher level of anonymity, meaning that a key is used a minimal number of times before switching to a new key. To accomplish this goal, CASP provides keychains, known as hierarchical deterministic wallets (or "HD Wallets"). These keychains enable the creation of many keys for a single vault.

Working with the currency layer, CASP supports a specific hierarchy structure as defined in the BIP44 scheme. This scheme supports the following:

- Generating any number of sub-accounts within the same vault. You can use this approach for creating segregated accounts within the same vault.
- Generating many addresses for each sub-account.

This section describes the APIs for management of these keychains. If you require a different keychain structure, see the section on BYOW Keychain Management.

4.15. Operators

Click here to open the Operator APIs.

Operators manage users, participants, and vaults. They initiate operations and manage policies. An operator can be configured to have 2-factor authentication ("2FA"). If enabled, when the operator attempts to log into the web interface, an authentication request is sent to that user's mobile device. The user is only permitted to access the web interface after 2FA approval on the mobile device.

4.16. Participants

Click here to open the Participant APIs.

A participant can be a human within the account or a BOT taking part in crypto asset transactions. Each participant owns a share of the cryptographic material that is part of the different transactions. CASP supports participants using any relevant platform, including mobile devices, laptops, and different server platforms for BOT's.

4.17. Policy Management

To open the Policy Management APIs, click for Built-in Wallets or BYOW.

Each CASP vault has a set of risk-based quorum policies associated with it, which are defined during vault creation (see Risk-Based Policy Vaults for policy configuration in the Web UI). These policies assign a different quorum policy to different transactions, based on the transaction details (such as the transaction amount or the time of day).

Copyright 2021, Unbound Security

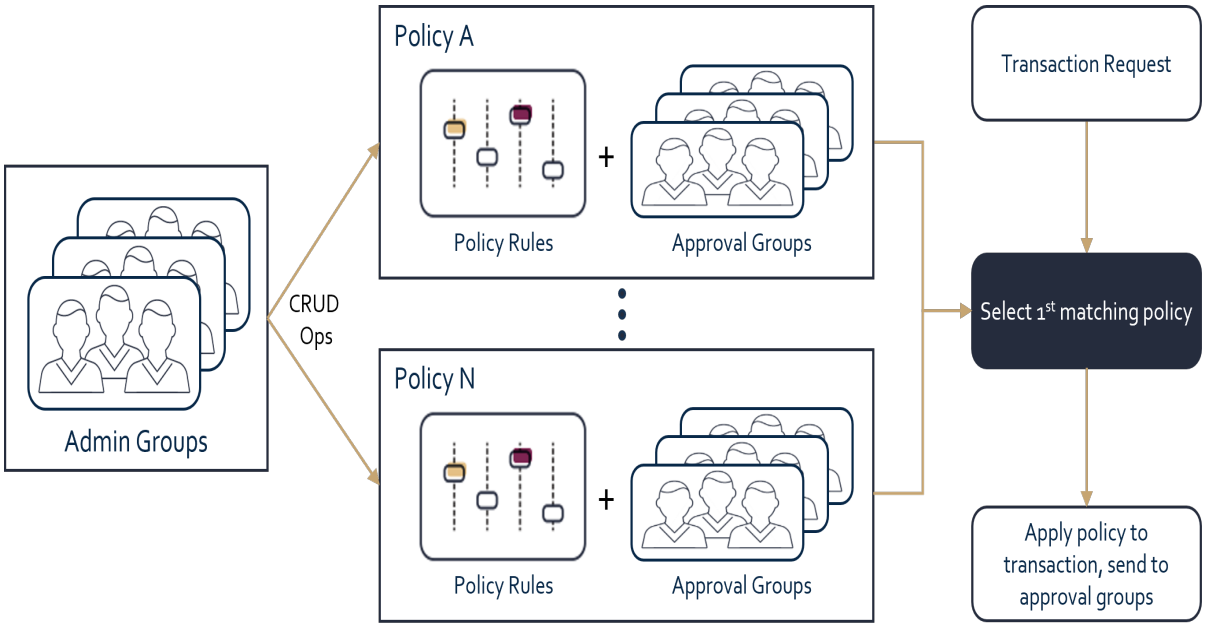
4.17.1. MofN Quorum Groups

Approvals are defined by a group of authorizing entities, of which a minimal-size subset (called a quorum) is required to approve the transaction. M approvals from a set of N entities is known as "MofN". For example, the client may define 8 entities, of which 4 must approve the transaction. Another example is where there must be 3 approvals from group A and 2 approvals from group B.

The number of groups, size of groups, and the size of the approving subset is fully flexible and can be different for each vault.

4.17.2. Policy Architecture

The following figure shows the architecture of the policy management components.



There are two types of quorum groups:

1. **Admin Groups** - quorum groups used to approve vault creation and subsequent policy updates.
2. **Approval Groups** - quorum groups used to approve operations.

The Security Officer (see CASP Roles) of the CASP system can create, read, update, and delete (CRUD) policies in a vault. These actions (except for read) require the approval of the admin groups defined for the vault.

Each **policy** is comprised of a set of policy rules and a set of approval groups. The policy rules define limits on the transaction amounts and restrictions on the time of day when the policy is active. You can create as many policies as required for your vault.

When a transaction, such as a withdrawal, is requested, CASP checks the list of policies to determine the first policy that has rules that match the conditions of the transaction. This policy is then applied to the policy resulting in a request to the approval groups associated with the policy.

Copyright 2021, Unbound Security

4.17.3. Whitelisting

Vaults can be created with a policy that contains rules that only apply for a *whitelist* or *derived whitelist*.

- A **whitelist** is a list of destination addresses for which the rule applies.
- A **derived whitelist** can whitelist all keys that can be derived from a specific key. The destination addresses must be able to be derived from the public key corresponding to the provided key.

The *whitelist* is specified in the endpoint for creating a new vault.

For example, the following is part of a request showing both a *whitelist* and *derived whitelist*:

```
"whiteList":
  [
    "F367002FFA7e97FFE5dd13855Ab5341Be18BC0fF"
  ],
"derivedWhiteList":
  [
    {
      "chainCode":
"6559FE7E680D021D715101D5B05193A5D0BDD11B55D9B949116EA0FD17268343",
      "publicKey": "3056301006072A8648CE3 ... E5CB1D1FAE95522A912F0E1E7CFB9CF",
      "level": 1,
      "parentFingerprint": "1811729397",
      "childNumber": 12
    }
  ],
```

When sending a sign request for this vault, there is a parameter called *derivedWhitelistChildNumbers* that specifies the child number. This parameter is needed when there are more than 1000 keys. For example: `"derivedWhitelistChildNumbers": [ 50008 ]`

It is important to note the following about **derived whitelists**:

1. Each *derivedWhiteList* item represents the **parent**.
2. The sign request parameter *derivedWhitelistChildNumbers* specifies the child number.
3. If *derivedWhitelistChildNumbers* is **not** specified, the first 1000 keys from the parent are derived (starting with *childNumber* 0 and ending with *childNumber* 999).
4. CASP checks that all the destination addresses in the transaction are either in the first 1000 derived children or they are change addresses.

For example, if we want to send money to key `m/42'/0'/2'/0/15`, we specify in the *derivedWhiteList* the BIP details of the parent key: `m/42'/0'/2'/0`. Furthermore, all keys from key `m/42'/0'/2'/0/0` to key `m/42'/0'/2'/0/999` would be accepted by this rule.

4.18. Reports

Click here to open the Report APIs.

Copyright 2021, Unbound Security

These endpoints provide reports about various aspects of the CASP system, such as vault details, user, details, and account details.

4.19. Trusted Systems

Trusted systems enable the exchange of data between separate CASP systems (i.e. systems that have an air gap) and enforce approval of transactions in these different systems. For example, they can be used to create a hot/cold environment where vault generation (and the key ceremony) occurs in the cold CASP system. Transactions are signed first by participants in the cold system and then by participants in the hot system.

4.20. Vault Operations

To open the Vault Operation APIs, click for Built-in Wallets or BYOW

A vault is a secure container for the cryptographic material used to protect a crypto asset, such as the seed or private key. CASP uses Multiparty Computation (MPC) to split the crypto material between the different participants in the vault, which ensures that the material never exists in a single place. In addition, only the approved set of participants can complete a transaction based on the vault definition.

A **Quorum** vault shares the responsibility of executing a transaction between many different participants in a structure defined by the vault policy. The vault policy contains a quorum-based structure where there are any number of groups, any threshold value per group, any tree structure between different groups, etc. The MPC protocols used by CASP ensure that if and only if the quorum definition is satisfied, a transaction can take place, which is enforced on the cryptographic level.

Use the relevant endpoint based on BYOW or the currency you want to work with. The list of supported currencies and the matching short names can be found in Supported Wallets. In this document, Bitcoin is used as an example.

The main endpoints for vault management are:

```
/btctest/api/v1.0/accounts/{accountId}/vaults
```

and

```
/btctest/api/v1.0/vaults/{vaultId}
```


Copyright 2021, Unbound Security

5. API Flows

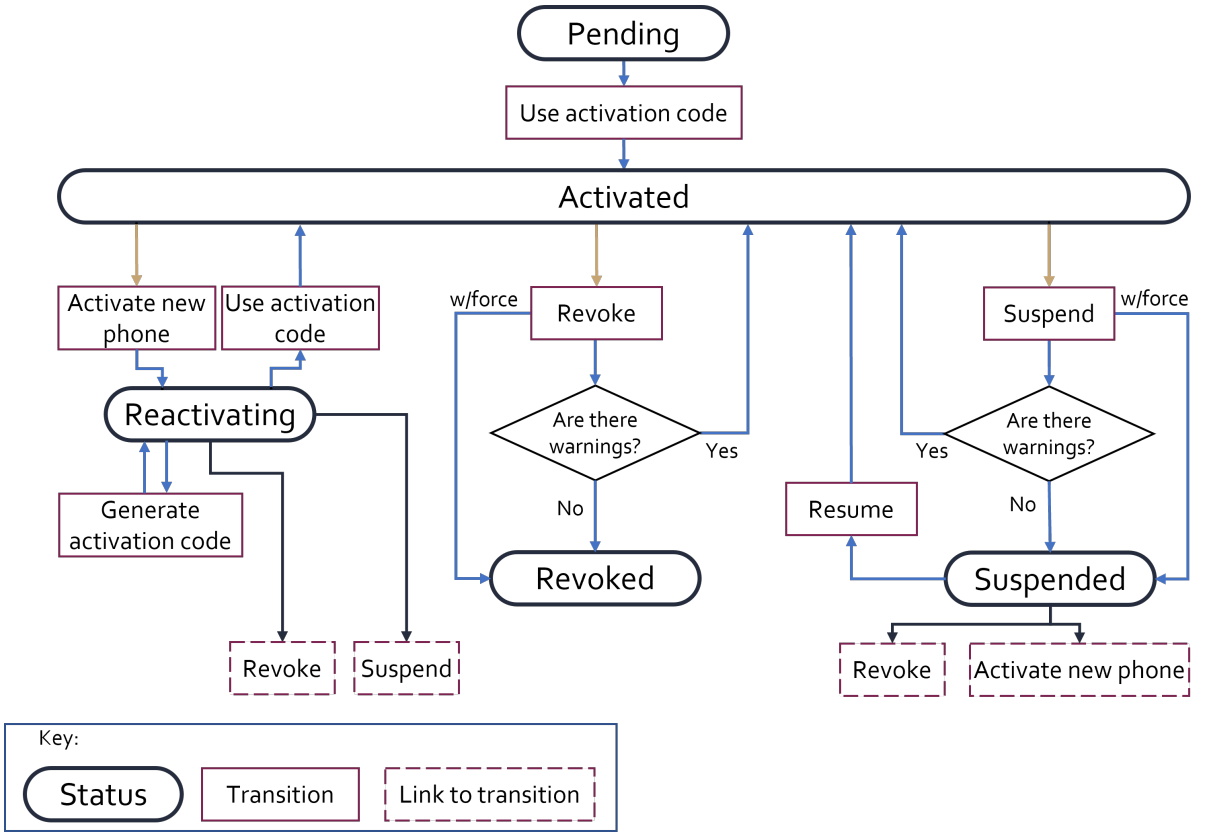
Descriptions of various API flows are provided in the following sections.

5.1. Participant Status

Participants can have one of these statuses:

- Pending - initial state for users in the CASP system.
- Activated - the participant can approve/deny operations.
- Suspended - the participant **temporarily** cannot approve/deny any operations.
- Revoked - the participant **permanently** cannot approve/deny any operations.
- Reactivating - used to move a participant to a new phone. The participant remains in this state until use of the activation code.

The following figure shows the possible statuses along with the actions that can be used to change the status.



Note
The **Revoked** status is permanent. There is no method to move from Revoked status to any other status.

The following sections describe operations that affect the participant status.

Copyright 2021, Unbound Security

Tip
An endpoint, called Participant change warnings, is provided to check if there are any warnings associated with a potential change in status.
For example, if you suspend a participant, a quorum may drop below the threshold required to approve operations. Therefore, you may want to add another participant to the quorum before suspending the participant.

Warning
Special care must be taken when changing the status of a participant. If you use the *force* flag, you can cause a vault to become unusable.
For example, if you have a simple policy vault with 3 required participants for approvals and you revoke one of them, you can no longer approve **any** transactions for the vault.

5.1.1. Activate new phone (device)

If a participant loses a phone (or whatever device the participant is running on) or gets a new phone, the participant needs to be reactivated. Reactivation includes activating a participant and re-adding to all relevant vaults.

If there were any pending operations to join a vault, either for the participant to approve or the admin to approve, the reactivation fails.

- **Scope**: Global
- **Permanent**: No
- **Approval**: Yes, for each vault
- **APIs**:
  - Reactivate with Reactivate a participant.

5.1.2. Global Suspend

Temporarily remove a participant from all policies.

If suspending violates the quorum threshold, the suspend fails. A flag is provided to force the suspend operation.

- **Scope**: Global
- **Permanent**: No
- **Approval**: None
- **APIs**:
  - Get the relevant groups with Participant group info.
  - Change the status with Update an existing participant.

5.1.3. Local Suspend

Temporarily remove a participant from a specific quorum or a simple policy vault.

Copyright 2021, Unbound Security

If suspending violates the quorum threshold or if the participant has pending approval operations, the suspend fails. A flag is provided to force the suspend operation.

- **Scope**: Specific quorum or a simple policy vault
- **Permanent**: No
- **Approval**: Admin group
- **APIs**:
  - Suspend a vault member with Set vault member status.
  - Suspend a policy member with Set policy member status.

5.1.4. Global Resume

Globally restores a participant that was suspended.

If a participant was suspended from a specific vault, the participant will remain suspended.

- **Scope**: Global
- **Permanent**: No
- **Approval**: None
- **APIs**:
  - Change the status with Update an existing participant.

5.1.5. Local Resume

Restore a participant that was suspended from a specific quorum or a simple policy vault.

- **Scope**: Specific quorum or a simple policy vault
- **Permanent**: No
- **Approval**: Admin group
- **APIs**:
  - Resume a vault member with Set vault member status.
  - Resume a policy member with Set policy member status.

5.1.6. Revoke

Permanently remove a participant from all policies or from a specific policy.

If revoking violates the quorum threshold or if the participant has pending approval operation, the revoke fails. A flag is provided to force the revoke operation.

- **Scope**: Global
- **Permanent**: Yes
- **Approval**: Admin group
- **APIs**:
  - Change the status with Update an existing participant.

Copyright 2021, Unbound Security

5.2. Participant Flows

CASP provides APIs for these participant flows:

1. Create a participant
2. Update participant details
3. Create a vault with existing participants
4. Add a participant to an existing vault
5. Participant put on hold globally or in a specific vault (suspend)
6. Participant leaves the company or a specific vault (revoke)
7. Participant replaces a phone (reactivate)
8. BOT becomes unavailable

The flows are described in the following sections.

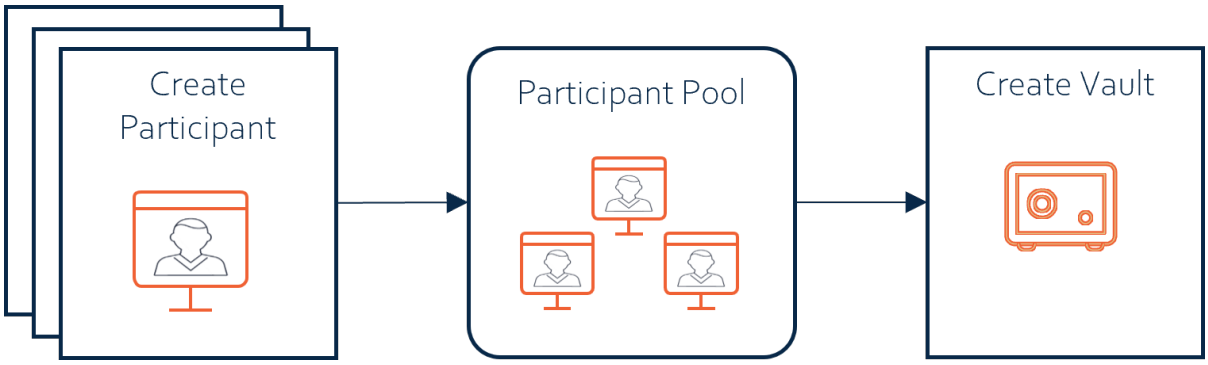
5.2.1. Create a participant

A participant is created in a specific account with the Create a new participant endpoint.

5.2.2. Update participant details

You can update a participant's details, including the name and email address, with the Update an existing participant endpoint.

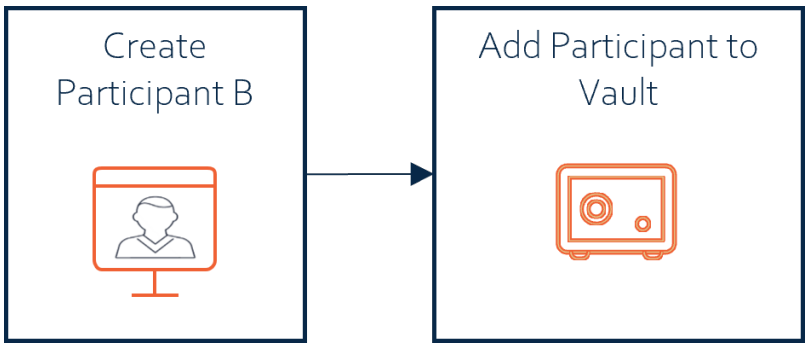
5.2.3. Create a vault with existing participants



- Step 1: Create participants. You need enough activated participants to implement the quorum groups for your vault.
- Step 2: Create a vault using **Create a new vault** (in Built-in or BYOW). During vault creation, you assign participants to be members of the relevant quorum groups.
- Step 3: Each quorum group member needs to approve joining the vault.

Copyright 2021, Unbound Security

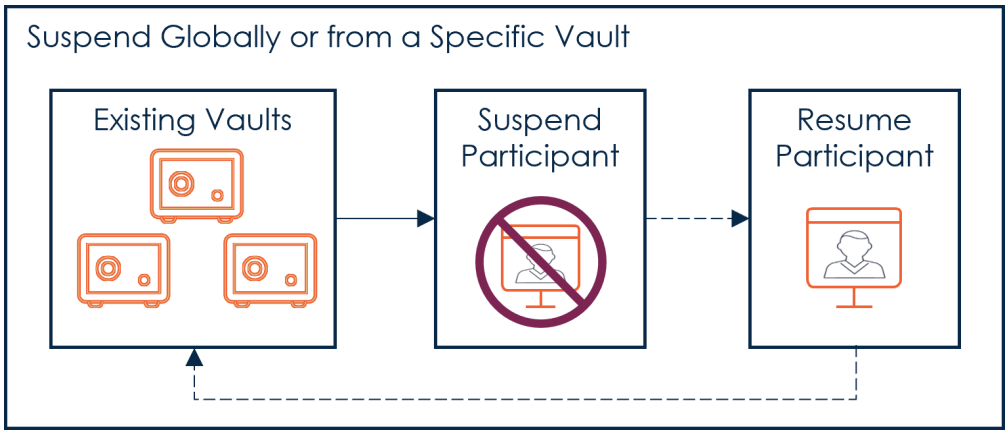
5.2.4. Add a participant to an existing vault



- Step 1: Create and activate a new participant (Participant B) if the participant does not exist.
- Step 2: Add this participant to one of the quorum groups in the vault with **Add vault member** (in Built-in or BYOW).
- Step 3: The required quorum group members must approve this operation.
- Step 4: The participant must approve being added to the vault.

5.2.5. Participant put on hold globally or in a specific vault (suspend)

If you want to temporarily remove a participant from a vault's quorum groups, you can Global Suspend or Local Suspend the participant. Suspending a participant is a **temporary** action and is done either at the vault level or globally, using Update an existing participant. After suspending, the participant can no longer approve operations.



A suspended participant can be resumed so that they can approve/deny operations. Endpoints are provided to Global Resume or Local Resume the participant.

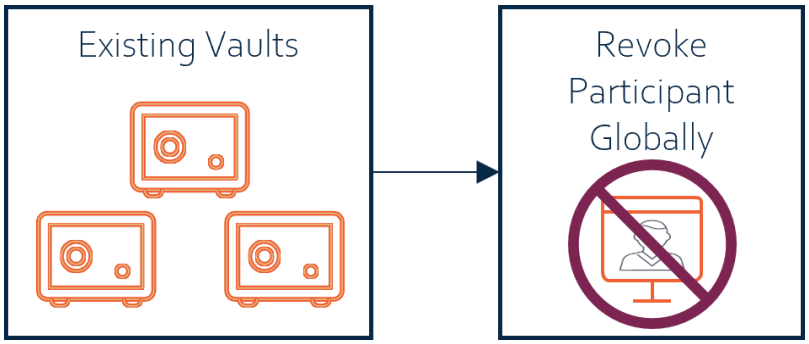
5.2.6. Participant leaves the company or a specific vault (revoke)

If you no longer want a participant to be a part of the vault's quorum groups, you can Revoke the participant. Revoking a participant is a **permanent** action and is done either at

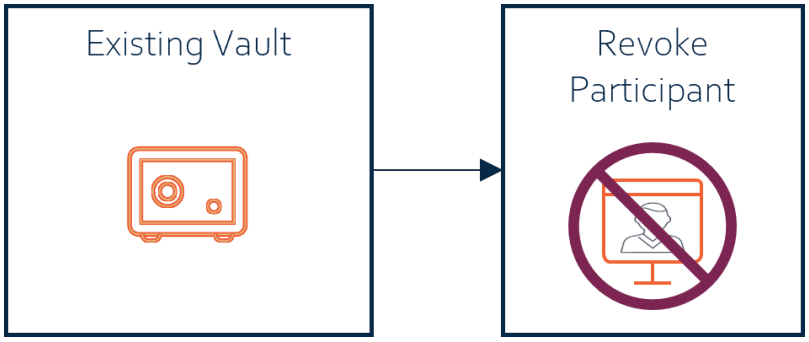
Copyright 2021, Unbound Security

the vault level or globally, using Update an existing participant. After revoking, the participant can no longer approve operations.

1. **Revoke globally**



2. **Revoke from a single vault**



Warning
Revoking a participant is permanent and cannot be undone.

5.2.7. Participant replaces a phone (reactivate)

If a participant replaces their phone (such as if it was lost and then they buy a new one), the participant needs to be reactivated with Reactivate a participant. This endpoint activates the participant and executes a re-add operation for each relevant vault. The participant needs to be approved by the admin group in each vault.

Notes
1. Participants cannot approve/deny operations until the re-add operation has been approved.
2. The participant's old phone continues to work until the participant activates the new phone. If a phone is lost, it is recommended to immediately suspend the participant to prevent the old device from being used.

Note
When a participant is activated, a unique key share is created by the app. If this key share is in any way removed, the participant needs to be reactivated (to create a new key share) and then re-added to any vaults. Key share loss is a result of **any** of these actions:
- The app is removed from the device.
- The user resets the app data.
- The user gets a new phone.

Copyright 2021, Unbound Security

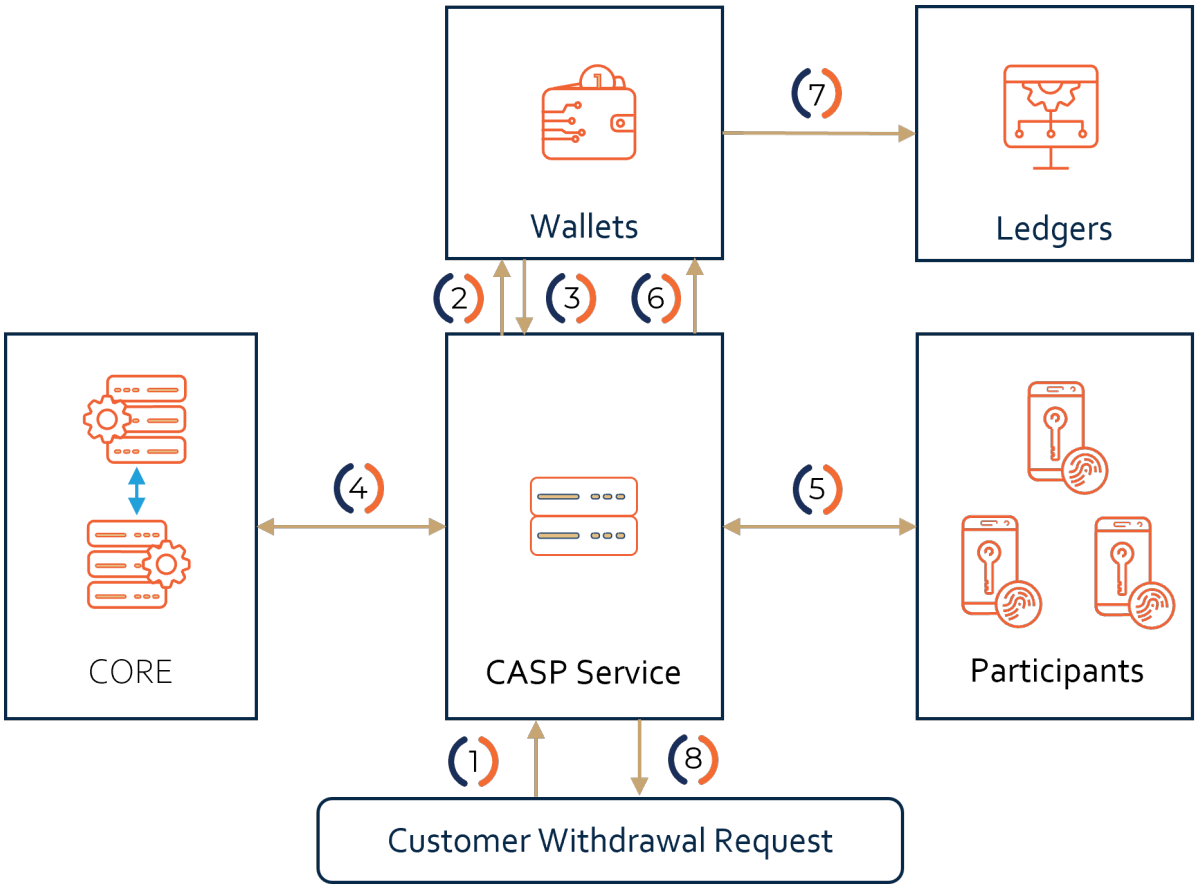
5.2.8. BOT becomes unavailable

If one of your BOT's data is lost or otherwise unavailable, you can replace it with these steps:

- Step 1: Revoke the BOT globally.
- Step 2: Run the steps in Add a participant to an existing vault to activate the new BOT in all of the relevant vaults.

5.3. Transaction Signing Flow

The following description details how CASP signs a transaction. It assumes that CASP is already installed and the participants have been created and activated. Each participant received a key share that it uses for signing.



Note
In the figure, the *Wallets* block is either a wallet using one of the built-in CASP wallets or BYOW. In the following steps, if one is using a built-in wallet, then all the functionality is built into CASP. BYOW wallet implementations need to implement the associated functions as well.

Copyright 2021, Unbound Security

1. The customer/client asks to move some amount of crypto assets to a target address. A withdrawal request is sent to CASP that includes the raw transaction details and the hash of the raw transaction based on the specific protocol. Use the endpoint Start withdrawal process (for built-in wallets) or Start a signing process (for BYOW).
2. The wallet computes the relevant transaction details after reading info from the ledger verifying balances, etc.
3. The transaction and hash are sent to the CASP service for signing.
4. Partial work is done on CASP and CORE.
5. The transaction details and the partial interim results are sent to all participants. The participants execute their part and send their results back to the CASP server.
6. Once enough results are collected, the signature is computed and returned to the wallet.
7. The wallet sends the signed transaction to the ledger.
8. The plain signature is returned to the customer as a confirmation of the withdrawal.

At any step, you can check on the status of the signing process with Get withdrawal information (for built-in wallets) or Get sign operation (for BYOW).

Copyright 2021, Unbound Security

6. API Reference

CASP APIs are provided in these three links:

1. CASP Base APIs
   - Authentication
   - Attributes
   - Accounts
   - Operators
   - Participants
   - Trusted Systems
   - Reports
   - Backup
   - Audit
2. CASP Built-in Wallet APIs
   - System
   - Ledger Operations
   - Vault Operations
   - Policy Management
3. CASP BYOW APIs
   - BYOW Vault Operations
   - BYOW Policy Management
   - BYOW Keychain Management

Copyright 2021, Unbound Security

7. CASP Java SDK

The CASP Java SDK enables:

1. Integration of the CASP approval process into your applications.
2. Management of CASP participant data.
3. Handling data for approving CASP transactions.
4. Data collector functionality.

The CASP Java SDK, alongside the CASP Server, supports storing and managing CASP vaults using any client device.

7.1. Integrating CASP with Client Applications

As explained in the Overview, the CASP service enables the creation of crypto asset vaults, where the management responsibility is shared by a quorum of participants. The role of a participant is implemented using the **CASP Java SDK**. The SDK provides functionality to create participants and data collectors.

The CASP Java SDK enables integration of the CASP approval process into your applications, management of CASP participant data, handling data for approving CASP transactions, and data collector functions. The CASP Java SDK, alongside the CASP Server, supports storing and managing CASP vaults using any client device.

7.1.1. Participants

A quorum vault shares the responsibility of executing a transaction between many different participants in a structure defined by the vault policy. In this case, the client plays the role of one participant in the quorum.

The CASP Java SDK supports the following high-level operations:

1. Initial setup and registration with the CASP service.
2. Participation in a quorum vault and the quorum approval process.
3. Query and track asynchronous operations where the client is a participant, and allow execution of the relevant operations.

The CASP Java SDK does not provide the storage or key management interfaces, but reference code is provided in CASP Java SDK Interfaces.

The technology for communicating with the CASP service as a participant is a REST API, so any platform can be supported.

The following sections include detailed descriptions for the SDK of the different supported clients. If you require integration with a different platform for a CASP participant, contact Unbound Support.

The CASP Java SDK supports offline participants who are disconnected from the network. These participants receive all data through manual transfer (such as using a USB flash drive). See Offline Vaults for more information.

Copyright 2021, Unbound Security

If you are using **offline participants**, first obtain the normal package, as described in Package Details, then for implementation details see Offline Usage.

7.1.2. Data Collectors

The CASP Java SDK also provides functionality to create **Data Collectors**. Data Collectors empower customers to add custom dynamic attributes to a CASP vault using Data Collector bots. Data Collectors are independent components that collect data from the organization's satellite systems, such as AML, KYC, Pricing, and Tax. Upon collection of enough data, CASP applies a predefined approval policy that considers the dynamic data that was brought externally to CASP.

CASP applies multi-party, MPC-based, approval policies to various types of operations and asset types. The variety of asset types and the attributes that can be associated with them requires CASP to stay asset-agnostic on one hand while keeping the highest security level on the other hand.

Placing the utmost importance on security while maintaining a commitment to being asset agnostic, led Unbound to add the **Data Collectors** capability.

Data Collectors empower customers to add custom dynamic attributes to a CASP vault using Data Collector bots. Data Collectors are independent components that collect data from the organization's satellite systems, such as AML, KYC, Pricing, and Tax. Upon collection of enough data, CASP applies a predefined approval policy that considers the dynamic data that was brought externally to CASP.

Allowing CASP to consume dynamic data that Data Collectors bring in externally to CASP, a customer must define an **Attribute Template**. An Attribute Template defines the attribute type (Boolean, date, numeric or string) and the range of values that the template can be associated with. For example, for an AML grade that comes in from a CipherTrace data collector, the attribute type would be numeric, and the range of values can be between 0-10.

Each data collector is associated with an attribute template group. The underlying assumption is that a data collector collects several types of attributes that are tightly associated with each other, application or process-wise. For example, KYC attributes, such as customer name, ID proof, address proof and picture proof, could be grouped under a KYC attribute template group.

In CASP, the workflow is that you first define an **attribute template**. You then group these templates into **attribute template groups**. Finally, you create a **data collector** that provides the data for these attributes. You can include data collectors in the vault policy when creating a new vault.

7.2. Prerequisites

The CASP Java SDK can be used on these operating systems:

- Ubuntu 16.04
- CentOS 7.7

Copyright 2021, Unbound Security

7.3. Package Details

Download and set up the **CASP Java SDK** package.

1. Access the CASP repository from the link provided to you by Unbound.
2. Select *client*, then the operating system (such as *centos)*, and then download the corresponding *casp-sdk-package.{version}.tar.gz* file.
3. Decompress the package.

The Java package contains the following:

- bin - libraries that are dependencies for the Java files.
- docs - HTML descriptions of the APIs.
- jar - contains the CASP Java SDK Java code.
- bot-sample - sample code for a bot participant.
- dc-sample - sample code for a data collector.

7.4. Developing Participants

The following sections describe how to create a CASP participant.

7.4.1. SDK Entry Point

The main entry to the CASP Java SDK for Java is the class:

- com.unboundtech.casp.android.sdk.CaspSdk

Use its instance function to get a singleton instance to work with:

```
public static CaspSdk instance()
```

It returns the instance of the SDK.

Example:

```
CaspSdk casp = CaspSdk.instance();
```

7.4.2. Initialization

```
CaspKeyManager caspKeyManager = new // implementation of CaspKeyManager
CaspStorage caspStorage = new //implementation of CaspStorage

JavaRestClient restClient = new JavaRestClient(serverUrl,
allowInsecureConnection, caspKeyManager, null);
CaspStatus initStatus = new CaspInitBuilder(caspStorage, caspKeyManager)
    restClient(restClient)
    init();
```

- Replace *serverUrl* with the URL of the CASP server.
- You must check the return status to determine if this initialized successfully.

Copyright 2021, Unbound Security

7.4.3. Activate Participant

To create a new participant, the participant must first be created on the CASP server. After creation, a participant ID and an activation code must be provided to the participant using the CASP Java SDK. This operation needs to be done once. After activation is completed, the participant can join vaults and approve operations.

To activate the participant:

```
public void activateParticipant(String participantId,
String activationCode,
String pushToken,
CaspSdk.CompletionListener listener)

CaspSdk.instance().activateParticipant(participantID, activationCode, null,
deviceType , status -> {
  if (status.getCode() != 0) {
    System.out.println("Could not activate participant. code: " + status.getCode
() + " msg: " + status.getDescription());
  } else {
    System.out.println("Participant successfully activated");
  }
});
```

Parameters:

- *participantId* - the participant ID.
- *activationCode* - the activation code.
- listener - completion listener.

7.4.4. Get Participant Information

After a participant is activated you can get the relevant information from the CASP server.

```
public void getParticipant(CaspSdk.GetParticipantListener listener)
```

Parameters:

- listener - completion listener

7.4.5. List Operations

An operation is triggered on the CASP server and then a corresponding operation is created for the relevant participant. The CASP Java SDK provides APIs for retrieving the number of pending operations and the list of operations for the specific participant.

To receive the number of pending operations:

```
public void getOperationsCount(CaspSdk.OperationsCountListener listener)
```

Parameters:

- listener - completion listener

Copyright 2021, Unbound Security

To receive a list of pending operations:

```
public void listOperations(CaspSdk.OperationsLoadedListener listener)
```

Parameters:

- listener - completion listener

For example:

```
CaspSdk.instance().listOperations((status, operations) -> {
  if (status.getCode() != 0) {
    System.out.println("Failed to list operations from CASP");
    latch1.countDown();
    return;
  }
  if (operations.size() == 0) {
    latch1.countDown();
    return;
  }
  System.out.println("There are " + operations.size() + " pending operations");
  handleOperations(operations);
  latch1.countDown();
});
```

7.4.6. Count Operations

```
CaspSdk.instance().operationsCount((status1, count) -> {
  //do stuff
);
```

7.4.7. Join Vaults

Join quorum vault

```
if (operation.kind == CaspOperation.Kind.JOIN_VAULT) {
  CaspSdk.instance().joinVault(operation, status -> {
    //stuff
  });
}
```

Join admin quorum

```
if (operation.kind == CaspOperation.Kind.JOIN_ADMIN_QUORUM) {
CaspSdk.instance().epJoinAdminQuorumVault(operation, status -> {
//stuff
});
}
```

Join risk-based policy vault

```
if (operation.kind == CaspOperation.Kind.JOIN_POLICY_VAULT) {
  CaspSdk.instance().epJoinPolicyVault(operation, status -> {
    //stuff
  });
}
```


Copyright 2021, Unbound Security

7.4.8. Approve Operations

- Transaction signing approval
- Add a participant to a vault approval
- Admin quorum approval

```
CaspSdk.instance().approveOperation(operation, status -> {
  //stuff
});
```

7.4.9. Decline Operation

```
CaspSdk.instance().declineOperation(operation, (CaspStatus status) -> {
  //stuff
});
```

7.4.10. Offline Usage

An offline vault is a vault that has some participants that are disconnected from the network. These participants receive all data through manual transfer (such as using a USB flash drive). Offline vaults enable customers to create solutions where part of the approval process takes places in highly secure areas, such as keeping the offline participants in a guarded vault.

The process for handling offline participants is described in the following sections.

7.4.10.1. Participant Activation

For participant activation:

1. Create an offline participant on the CASP system. A JSON file is downloaded.
2. Manually transfer the JSON file to the offline participant's device.
3. On the offline device, activate the offline participant. Save the response into a file (e.g. *activation_response.json*).
4. Manually transfer the file (*activation_response.json*).
5. Send the activation response to the CASP server to complete the activation.

The participant is now active.

Initialization

```
CaspKeyManager caspKeyManager = new // implementation of CaspKeyManager
CaspStorage caspStorage = new //implementation of CaspStorage

CaspStatus initStatus = new CaspInitBuilder(caspStorage, caspKeyManager)
    init();
```

- You must check the return status to determine if this initialized successfully.

Copyright 2021, Unbound Security

SDK Entry Point

The main entry to the CASP Java SDK for Java is the class **com.unboundtech.casp.android.sdk.CaspSdk**.

Use its instance function to get a singleton instance to work with:

```
public static CaspSdk instance()
```

Returns:

- The instance of the SDK

Example:

```
CaspSdk casp = CaspSdk.instance();
```

Activate Participant API

To create a new participant, the participant must first be created on the CASP server. After creation, a participant ID and an activation code must be provided to the participant using the CASP Java SDK. This operation needs to be done once. After activation is completed, the participant can join vaults and approve operations.

To activate the participant:

```
CaspSdk.instance().activateOfflineParticipant(activationRequestString,
activationCode, participantID, CaspSdk.DeviceType.BOT, (status,
offlineActivationResponse) -> {
  // check status
});
```

- *activationRequestString* - a Java string containing the content of the offline participant activation request that the server prepares when an offline participant is created.
- Replace *participantID* with the provided participant ID.
- Replace *activationCode* with the provided activation code.
- *offlineActivationResponse* – this response must be saved to a file (e.g. *activation_response.json*) and transferred back to the CASP server.

7.4.10.2. Operation Approval

For operation approval:

1. Start an operation on the CASP system.
2. Retrieve the offline participant operations. Place the response in a file (e.g. *operations_request.json*).
3. Manually transfer the JSON file to the participant's device.
4. Approve the operation on the participant's device. Save the response into a file (e.g. *operations_response.json*).
5. Manually transfer the file (*operations_response.json*).
6. Send the operation results to the CASP server to complete the operation.

Copyright 2021, Unbound Security

List Operations API

An operation is triggered on the CASP server and then a corresponding operation is created for the relevant participant. The CASP Java SDK provides APIs for retrieving the number of pending operations and the list of operations for the specific participant.

To receive the number of pending operations:

```
CaspSdk.instance().listOfflineOperations(operationsRequestString, new
CaspSdk.OperationsLoadedListener() {
  @Override
  public void done(CaspStatus status, List<CaspOperation> operations) {
    // Other operations
  }
}
```

- *operationsRequestString* - a Java string containing the content of the operation request that the server prepares when an operation is created.

Approve Operations API

Use this API to approve all operations passed into it.

```
CaspSdk.instance().approveOfflineOperations(approvedOperations, new
CaspSdk.OperationsApprovedListener(){
@Override
public void done(CaspStatus status, String encryptedOperationResults) {
    // Write results to file
  }
}
```

- *operationsRequestString* - a Java string containing the content of the operation request that the server prepares when an operation is created.
- *encryptedOperationResults* - a Java string containing the results of the operation request. It must be written to a file and transferred back to the CASP server.

Decline Operation API

There is no offline version of "decline".

7.5. Developing Data Collectors

The following sections describe how to develop a Data Collector.

7.5.1. SDK Entry Point

The main entry to the CASP Java SDK is the class:

```
com.unboundtech.casp.desktop.dc. DataCollectorSdk
```

Use its instance function to get a singleton instance to work with:

```
public static DataCollectorSdk getInstance()
```

It returns the instance of the SDK.

Example:

Copyright 2021, Unbound Security

```
DataCollectorSdk dataCollectorSdk = DataCollectorSdk.getInstance();
```

7.5.2. Initialization

```
CaspKeyManager caspKeyManager = new // implementation of CaspKeyManager
CaspStorage caspStorage = new //implementation of CaspStorage

JavaRestClient restClient = new JavaRestClient(serverUrl,
allowInsecureConnection, caspKeyManager, null);
CaspStatus initStatus = new DataCollectorSdkInitBuilder(keyStoreStorage,
keyStoreStorage, javaRestClient)

.init();
```

- Replace *serverUrl* with the URL of the CASP server.
- You must check the return status to determine if the SDK initialized successfully.

7.5.3. Activate Data Collector

To create a new data collector, the data collector must first be created on the CASP server. After creation, a data collector ID and an activation code must be provided to the data collector using the CASP Java SDK. This operation needs to be done once. After activation is completed, the data collector can provide data to operations.

To activate the data collector:

```
DataCollectorSdk.getInstance().activateDataCollector(dcId, activationCode, status
-> {
  if (status.getCode() == CaspStatus.DY_SUCCESS) {
    System.out.println("activation successful");
  } else {
    System.err.println("DC activation failed. " + status.getDescription());
  }
});
```

Parameters:

- dcId - the data collector ID.
- activationCode - the activation code.
- status - completion status.

7.5.4. Get Data Collection Request

After a data collector is activated you can get the relevant information from the CASP server.

```
public void getDataCollectionRequest(DataCollectionRequestListener listener)
```

Parameters:

- listener - completion listener.

The **DataCollectionRequestListener** listener provides the following information:

Copyright 2021, Unbound Security

```
public interface DataCollectionRequestListener {
  void done(CaspStatus status, DataCollection dataCollectionRequest);
}
```

The status variable tells if there are pending data collection requests for this DC.

Possible values:

- CaspStatus.DY_ENO_ENTITY – no pending data collection requests.
- CaspStatus.DY_SUCCESS – there is a pending data collection request. Check the **dataCollectionRequest** variable.
- Any other return code signifies an error.

Example:

```
DataCollectorSdk.getInstance().getDataCollectionRequest
((getDataCollectionRequestStatus, dataCollectionRequest) -> {
  if (getDataCollectionRequestStatus.getCode() == CaspStatus.DY_ENO_ENTITY) {
    System.out.println("no data collection request");
    latch.countDown();
    return;
  }
  if (getDataCollectionRequestStatus.getCode() != 0) {
    System.err.println("failed to retrieve data collection request. " +
getDataCollectionRequestStatus.getDescription());
    return;
  }
  System.out.println("data collection details: " + dataCollectionRequest.toString
());
});
```

The content of a **DataCollection** plain old Java object ("POJO"):

```
/**
  * Represents a data collection request from the CASP service. Use the {@link
DataCollection#collectData(Map, CaspSdk.CompletionListener)} to provide the
actual data.
  */
public interface DataCollection {
  /**
    * @return vault ID
    */
  String getVaultId();
  /**
    * @return vault name
    */
  String getVaultName();
  /**
    * @return sign request exactly as it was received by the CASP service
    */
  String getSignRequest();
  /**
    * @return the operation ID
    */
  String getOperationId();
  /**
    * @return the time at which the sign request was received
```


Copyright 2021, Unbound Security

```
    */
  long getOperationCreationTime();
  /**
    * @return the attributes that the Data Collector needs to provide
    */
  List<String> getAttributes();
  /**
    * Sends the {@code data} to the CASP service to be used in policy resolution.
<br>
    * If {@code data} holds more attributes than are specified in {@link
DataCollection#getAttributes}, the extra attributes are ignored.<br>
    * If {@code data} does not provide data for all attributes that are specified
in {@link DataCollection#getAttributes}, CASP rejects the data.
    *
    * @param data the collected data
    * @param listener completion listener with {@link CaspStatus}
    */
  void collectData(Map<String, String> data, CaspSdk.CompletionListener
listener);
}
```

7.5.5. Collect Data

If there is a pending data collection request, a data collector can provide data.

Assuming a data collection request is available (see Get Data Collection Request), data can be provided, such as:

```
Map<String, String> data = the data to provide
dataCollectionRequest.collectData(data, dataCollectionStatus -> {
  if (dataCollectionStatus.getCode() != 0) {
    System.err.println("failed to provide data. " +
dataCollectionStatus.getDescription());
  } else {
    System.out.println("Successfully provided data");
  }
});
```

7.6. CASP Java SDK Interfaces

You are required to provide the CASP Participant SDK with storage and key management interfaces. This section contains information and code samples to help you develop those interfaces.

7.6.1. Storage Interface

The storage interface that needs to be provided is *com.unboundtech.casp.desktop.signer.CaspStorage*. This interface enables the CASP Participant SDK to have persistent data and to access it as needed.

Instantiate the interface with this code:

```
public interface CaspStorage {
  void storeData(String key, String value) throws StorageException;
  String loadData(String key) throws StorageException;
```


Copyright 2021, Unbound Security

```
  List<String> listKeys() throws StorageException;
}
```

7.6.2. Key Manager Interface

The key manager interface needs these associated functions:

- generateEncryptionKey
- generateAuthenticationKey
- signWithAuthenticationKey
- decryptWithEncryptionKey

Instantiate the interface with this code:

```
public interface CaspKeyManager {
  void generateEncryptionKey(KeyReadyListener keyReadyListener);
  void generateAuthenticationKey(KeyReadyListener keyReadyListener);
  void signWithAuthenticationKey(AuthenticationStrength authenticationStrength,
byte[] data, String message, DataReadyListener
dataReadyListener);
  void decryptWithEncryptionKey(byte[] cypherText, DataReadyListener
dataReadyListener);
  CaspStatus deleteKeys();
}

public interface KeyReadyListener {
  void done(final CaspStatus status, final String key);
}

public enum AuthenticationStrength {
  WEAK, STRONG, NONE
}

public interface DataReadyListener {
  void done(final CaspStatus status, final byte[] data);
}
```

The Participant SDK uses the function **generateEncryptionKey** to generate the encryption key. This encryption key is used to decrypt data coming from the CASP server.

The **generateEncryptionKey** function:

1. Generates an RSA key-pair with a size of at least 2048-bits.
2. Returns the public key in PKCS#8 DER format, encoded in Base64.

For example:

```
@Override
public void generateEncryptionKey(KeyReadyListener keyReadyListener) {
  try {
    CertAndKeyGen certGen = new CertAndKeyGen(ENC_KEY_TYPE, ENCR_KEY_CERT_SIG);
    certGen.generate(ENC_KEY_SIZE);
    X509Certificate cert = certGen.getSelfCertificate(getX500Name(), FIVE_YEARS_
IN_SECONDS);
    this.keyStore.setKeyEntry(ENC_KEY_ALIAS, certGen.getPrivateKey(),
```


Copyright 2021, Unbound Security

```
password.toCharArray(), new X509Certificate[]{cert});
    saveKeystoreToDisk();
    keyReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_SUCCESS),
Base64Converter.toBase64(certGen.getPublicKey().getEncoded()));
  } catch (Exception e) {
    CaspLog.e(TAG, "failed to create encryption key", e);
    keyReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_EFAIL),
null);
  }
}
```

The function **generateAuthenticationKey** generates the authentication key, which is used to authenticate the participant to the CASP service.

The **generateAuthenticationKey** function:

1. Generates an EC key with a size of at least 256-bits.
2. Returns the public key in PKCS#8 DER format, encoded in Base64.

For example:

```
@Override
public void generateAuthenticationKey(KeyReadyListener keyReadyListener) {
  try {
    CertAndKeyGen certGen = new CertAndKeyGen(AUTH_KEY_TYPE, AUTH_KEY_CERT_SIG);
    certGen.generate(AUTH_KEY_SIZE);
    X509Certificate cert = null;
    cert = certGen.getSelfCertificate(getX500Name(), FIVE_YEARS_IN_SECONDS);
    this.keyStore.setKeyEntry(AUTH_KEY_ALIAS, certGen.getPrivateKey(),
password.toCharArray(), new X509Certificate[]{cert});
    saveKeystoreToDisk();
    keyReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_SUCCESS),
Base64Converter.toBase64(certGen.getPublicKey().getEncoded()));
  } catch (Exception e) {
    CaspLog.e(TAG, "failed to create authentication key", e);
    keyReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_EFAIL),
null);
  }
}
```

Notes
- **CertAndKeyGen** is part of the JDK.
- **bytesToBase64** is a method that gets a byte array as input and returns it encoded in base64 as string.

The **signWithAuthenticationKey** function does the following:

1. Signs the provided data.
2. Returns the signature in a PKCS#8 DER format, encoded in Base64.

For example:

```
@Override
public void signWithAuthenticationKey(AuthenticationStrength
authenticationStrength, byte[] dataToSign, String message, DataReadyListener
dataReadyListener) {
  try {
```


Copyright 2021, Unbound Security

```
    KeyStore.PrivateKeyEntry entry = (KeyStore.PrivateKeyEntry)
this.keyStore.getEntry(AUTH_KEY_ALIAS, getKeyStorePP());
    if (entry == null) {
      CaspLog.e(TAG, "auth key is missing");
      throw new CryptoException("authentication key missing");
    }
    Signature ecdsaSign = Signature.getInstance("SHA256withECDSA");
    ecdsaSign.initSign(entry.getPrivateKey());
    ecdsaSign.update(dataToSign);
    dataReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_SUCCESS),
ecdsaSign.sign());
  } catch (Exception e) {
    dataReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_EUSER_NOT_
AUTHENTICATED), null);
    CaspLog.e(TAG, "failed to sign with auth key", e);
  }
}
```

The **decryptWithEncryptionKey** function decrypts the participant data.

The function:

1. Decrypts that data with the cipher: `RSA/ECB/OAEPwithSHA-256andMGF1Padding`
2. Returns the plain text data.

For example:

```
@Override
public void decryptWithEncryptionKey(byte[] cypherText, DataReadyListener
dataReadyListener) {
  try {
    KeyStore.PrivateKeyEntry entry = (KeyStore.PrivateKeyEntry) keyStore.getEntry
(ENC_KEY_ALIAS, getKeyStorePP());
    if (entry == null) {
      CaspLog.e(TAG, "encryption key is missing");
      dataReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_EUSER_NOT_
AUTHENTICATED), null);
      throw new CryptoException("encryption key missing");
    }
    byte[] clearText;
    try {
      Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPwithSHA-
256andMGF1Padding");
      cipher.init(Cipher.DECRYPT_MODE, entry.getPrivateKey());
      clearText = cipher.doFinal(cypherText);
    } catch (BadPaddingException e) {
      Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1PADDING");
      cipher.init(Cipher.DECRYPT_MODE, entry.getPrivateKey());
      clearText = cipher.doFinal(cypherText);
    }
    dataReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_SUCCESS),
clearText);
  } catch (Exception e) {
    dataReadyListener.done(CaspStatus.constructByValue(CaspStatus.DY_EUSER_NOT_
AUTHENTICATED), null);
    CaspLog.e(TAG, "failed to sign with enc key", e);
```


Copyright 2021, Unbound Security

```
  }
}
```

7.6.3. REST Client

A REST client is provided as part of the SDK that can be used for production. The client assumes that you are using a direct connection (i.e. not a proxy).

The client can be instantiated as follows:

```
CaspRestClient restClient = new JavaRestClient(serverUrl,
allowInsecureConnection, caspKeyManager, null);
```

Note that you need to have *caspKeyManager* already initialized for this call.

The parameter *AuthenticationStrength* should always be set to STRONG.

```
public interface CaspRestClient {
  void post(AuthenticationStrength authenticationStrength, String endpoint,
String body, RestResponseListener listener);

  void put(AuthenticationStrength authenticationStrength, String endpoint, String
body, RestResponseListener listener);

  void get(AuthenticationStrength authenticationStrength, String endpoint,
RestResponseListener listener);

  void setResponseSignKey(String responseSignKey);

  interface RestResponseListener {
    void response(CaspStatus status, String body);
  }
}
```

7.6.4. Initialization

The CASP Participant SDK is initialized with the storage and key management interfaces.

```
CaspStatus initStatus = new CaspInitBuilder(caspStorage, caspKeyManager)
  restClient(restClient)
  init();
```

If you are using **offline** vaults, you can use this code to initialize:

```
CaspStatus initStatus = new CaspInitBuilder(caspStorage, caspKeyManager)
  init();
```


Copyright 2021, Unbound Security