



Unbound CORE Crypto Asset Security

User Guide

Version 1.0.2106
October 2021



Table of Contents

1. Revision History	1
2. Unbound CORE Overview	4
3. CORE CASP Solution Overview	6
3.1. CASP Components	6
3.1.1. Accounts	6
3.1.2. Participants	7
3.1.3. Vaults	7
3.1.4. Risk-Based Quorum Policy	7
3.1.5. Admin Quorum	7
3.1.6. Data Collectors	7
3.1.7. Synchronous and Asynchronous Operations	8
3.1.8. Blockchain	8
3.2. Using CASP to Sign a Transaction	8
3.3. Solution Architecture	8
4. System Requirements	10
4.1. Server Requirements	10
4.2. Client Requirements	11
4.3. Supported Browsers	12
4.4. Supported Wallets	12
4.5. System Limits	12
5. Installing CASP	13
5.1. Prerequisites	13
5.1.1. Install Java	13
5.1.2. Set up NodeJS	13
5.1.3. Set up Apache HTTP Server	14
5.1.4. Setup CORE	14
5.1.5. OpenSSL Configuration	16
5.2. CASP Package Installation	16
5.2.1. CASP Configuration	16
5.2.2. Configure the CASP Database	17
5.2.3. CASP Setup Utility	17
5.2.3.1. CORE Password Setup	18
5.2.3.2. CORE Health Check	18
5.2.3.3. Database Password Setup	19
5.2.3.4. Database Health Check	19

5.2.3.5. Firebase Token Setup	20
5.2.4. CASP Configuration File	20
5.2.5. Wallet Configuration	21
5.2.5.1. Ethereum and Ethereum Testnet	23
5.2.5.2. ERC-20	23
5.2.6. Start the CASP Services	24
5.2.7. Configure Apache	24
5.2.8. Configure Logs	25
5.2.9. Push Notifications (optional)	25
5.3. Testing the CASP System	26
5.4. Next Steps	27
6. Upgrading CASP	28
7. Single Sign-On	30
8. Reverse Proxy	33
9. Key Storage	34
10. Key Backup and Restore	35
10.1. Backup	35
10.2. Recovery Scenarios	37
10.2.1. Lost Phone, Damaged Bot, or Employee Replacement	37
10.2.2. CORE Loss	37
10.2.3. CASP Server Loss	37
10.2.4. CASP Database Loss	38
10.2.4.1. Total System Loss	38
10.3. CASP Restore Utility	38
10.3.1. Unbound Restore and Verify Utility	40
10.4. Master Key Extractor	40
10.5. Restore Examples	41
10.5.1. Restore a Deterministic (non-BIP) Vault	41
10.5.2. Restore an HD (BIP44) Vault	42
11. High Availability	44
11.1. CORE	44
11.2. CASP Database	45
11.3. CASP Orchestrator	45
11.4. Endpoints	45
12. Audit and Logging	46
12.1. CASP Log Files	46

12.2. CASP Built-in Wallet Logs	47
12.3. Configure Log4j	47
12.4. Connecting CASP to Syslog	47
12.5. CORE Log Files	50
12.6. Troubleshooting Audit Logs	50
12.6.1. 500 Error	50
13. Web Interface	52
13.1. CASP Initialization	52
13.1.1. Access the UI	52
13.1.2. Create an Account	52
13.1.3. Create a Vault	53
13.1.4. Add Participants	54
13.2. Accounts	54
13.3. Users	54
13.4. Participants	55
13.5. Vaults	57
13.5.1. Simple Vault	58
13.5.2. Risk-Based Policy Vault	59
13.5.3. Vault Backup	62
13.5.4. BIP32 and BIP44	62
13.6. Data Collectors	63
13.7. Templates	64
13.8. Operations	65
13.8.1. Transaction fee calculation	66
13.8.1.1. Bitcoin	66
13.8.1.2. Ethereum	67
13.9. Reports	67
13.10. System	69
13.11. Participant Management	70
13.11.1. Create a participant	70
13.11.2. Update participant details	70
13.11.3. Create a vault with existing participants	71
13.11.4. Add a participant to an existing vault	71
13.11.5. Participant put on hold globally or in a specific vault (suspend)	71
13.11.6. Participant leaves the company or a specific vault (revoke)	72
13.11.7. Participant replaces a phone (reactivate)	74

13.11.8. BOT becomes unavailable	75
13.11.9. Replace a Participant	75
14. Offline Vaults	76
14.1. Offline Participants	76
14.2. Offline Vaults	77
14.3. Offline Policies	78
15. Trusted Systems	79
15.1. Sample Trusted System Architecture	79
15.2. Configure Trusted Systems	79
15.2.1. Add a Trusted System	80
15.2.2. Synchronize Accounts	80
15.2.3. Synchronize Participants	81
15.2.4. Create and Synchronize Vaults	81
15.2.5. Synchronize Operations	82
15.3. Sign Transactions	82
15.4. (Optional) Synchronize CORE Keys	83
16. Mobile App	86
16.1. Prerequisites	86
16.2. App Setup	86
16.3. Participant Activation	88
16.4. Vault Approval	91
16.5. Operation Approval	97
16.6. 2-Factor Authentication	97
16.7. Settings	98
16.8. Profile	99
17. CASP Bot	101
17.1. System Requirements	101
17.2. Installation	101
17.3. Prerequisites	101
17.4. Activation	102
17.5. Enable Listening	102
17.6. Offline Bots	103
17.6.1. Create and Activate the Offline Bot	103
17.6.2. Add an Offline Bot to a Vault	105
17.6.3. Approve Transactions with the Offline Bot	105
17.7. Storing Keys in an External Security Provider	108

18. Troubleshooting	109
18.1. CASP Restore SO Password	110
18.2. Contact Support	110
19. Production Checklist	111
19.1. Installation	111
19.2. Vaults	111
19.3. Backup and restore	112
19.4. Security	112
19.5. Bot Security	112
Appendix A. PostgreSQL	113
A.1 Install PostgreSQL	113
A.2 Start PostgreSQL as a Service	113
A.3 Tweak PostgreSQL Permissions	113
Appendix B. Web UI Manual Installation	114
Appendix C. Using Databases Over SSL	117
C.1 RDS PostgreSQL Over SSL	117
C.1.1 Step 1: Create an RDS parameter group	117
C.1.2 Step 2: Update your RDS instance	117
C.1.3 Step 3: Configure PostgreSQL	117
C.1.4 Step 4: Configure SSL	118
C.1.5 Step 5: Configure CASP	118
C.1.6 Step 6: Check the SSL connection	118
C.2 RDS MySQL Over SSL	119
C.2.1 Step 1: Configure the certificate	119
C.2.2 Step 2: Configure CASP	119
C.2.3 Step 3: Check the SSL connection	120
Appendix D. Installing MySQL	121
Appendix E. Create a CSR for a CASP Vault	122

1. Revision History

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

Version	Date	Description
1.0.2106	October 2021	Updates from rebranding. Added Unbound CORE Overview . Added information about BIP32/44 to Key Backup and Restore .
1.0.2103	May 2021	Added section Single Sign-On . Updated CORE Password Setup with single sign-on details. Added section System Limits . Updated CASP Bot , removed the need to specify <code>libcasp_signer_jni.so</code> on the command line.
1.0.2010	January 2021	Added section Data Collectors . Added section on Attribute Templates . Added attribute information and rejection policy information to Risk-Based Policy Vault creation. Added section Troubleshooting Audit Logs .
1.0.2007	September 2020	Added Azure MySQL compatibility and IBM LinuxONE III support to System Requirements . Added an appendix to Create a CSR for a CASP Vault . Updated the script options in CASP Setup Utility and added a new section Firebase Token Setup . Added section CASP Database Cleanup Script . Added section Staking Vault Transactions .
1.0.2004	May 2020	Updated the Web Interface with a new look and feel. Added section on Offline Vaults . Added section on Trusted Systems . Updated Vaults with new information about offline vaults and trusted systems. Added section on Staking Policies . Updated the Mobile App with a secure proxy feature. Added Unbound Restore and Verify Utility .
1.0.2001	January 2020	Updated the section Wallet Configuration to include custom chain connectors. Added section on Reverse Proxy . Updated section Offline Bots . Added section Installing MySQL . Added section on supporting Ripple.
1.0.1910	December 2019	Added section on Transaction fee calculation .
1.0.1910	November 2019	Added section Offline Bots .
1.0.1907	October 2019	Added section OpenSSL Configuration . Updated section Push Notifications with new configuration script.

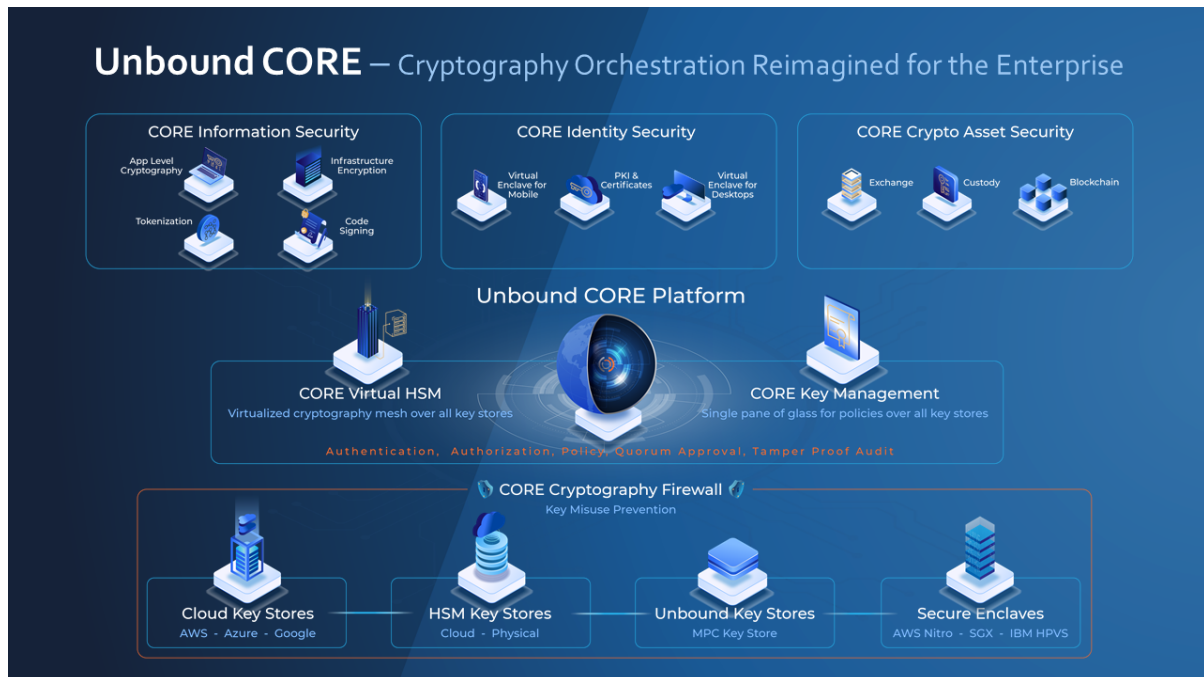
Version	Date	Description
		Updated section Built-in Wallet Configuration with new configuration script.
1.0.1906	September 2019	<p>Updated Participant Management with new status information.</p> <p>Updated Access the UI with new checkbox to remember the login for the session.</p> <p>Updated Web Interface with new user management screens.</p> <p>Updated Users section in the Web Interface with information about <i>Operators</i> and <i>Participants</i>, including 2-factor authentication.</p> <p>Updated Audit and Logging with new file format.</p> <p>Added missing flags in CASP Bot.</p> <p>Updated the commands/examples for MySQL to provide multilingual support in Configure the CASP Database and CASP Configuration File.</p>
1.0.1905	July 2019	<p>Added information about the CORE offline backup key script to Setup CORE.</p> <p>Added restoring to Electrum to the example in Restore Examples.</p> <p>Added Ubuntu installation instructions to the Installation section.</p> <p>Added section Upgrading from CASP 1.0.1810.</p>
1.0.1904	June 2019	<p>Added information about risk-based policy management to Risk-Based Vaults.</p> <p>Added section Audit and Logging.</p> <p>Added a missing prerequisite to PostgreSQL.</p> <p>Added information about the operation details in Operations.</p> <p>Added section Mobile App.</p> <p>Added section CASP Bot.</p> <p>The configuration instructions for Bitcoin and Bitcoin Testnet and Ethereum were updated.</p> <p>Added configuration for Push Notifications.</p> <p>Added section on Participant Management.</p> <p>Updated the minimum PostgreSQL version to 9.6 in System Requirements and PostgreSQL.</p> <p>Updated Start the CASP Services.</p> <p>Added section Mass Payouts.</p>
1.0.1902	March 2019	<p>Added section Vault Backup to the Web UI chapter.</p> <p>Added section Restore Examples.</p> <p>Added section CASP Restore Utility.</p> <p>Added section Production Checklist.</p>
1.0.1901	February 2019	Updated the Web UI installation instructions in Web Interface .
1.0.1812	February 2019	<p>Added section Web Interface.</p> <p>Added section Participant Management.</p> <p>Added new screen to generate an API key to System.</p> <p>Added section Master Key Extractor.</p> <p>Added section CASP Setup Utility.</p>

Version	Date	Description
		Added Bitcoin Cash to Wallet Configuration .
1.0.1811	January 2019	Rewrote the section on Key Backup and Restore . Added note about the new log file location in Wallet Configuration .
1.0.1811	December 2018	Added a note about changing the SSL certificate to Configure Apache . Updated the Ethereum description in Wallet Configuration with information about the fee settings. Added section on Upgrading CASP .
1.0.1810	December 2018	Added section Configure Logs .
1.0.1810	October 2018	Re-wrote the section on Installing CASP using the new RPM. Added Amazon Aurora and RDS to the support systems in System Requirements .
1.0.1809	September 2018	The document was updated for this 1809 release. All sections were modified.
1.0.1806	June 2018	Initial version.

2. Unbound CORE Overview

Unbound CORE protects and manages keys across your entire cryptographic infrastructure. No matter where you store your keys, physical HSMs, cloud HSMs, cloud key management systems, secure enclaves, CORE enables all applications to consume cryptographic services in the same way, anywhere, any cloud.

The CORE ecosystem is shown in the following image.



Unbound CORE is comprised of three components:

1. Unbound CORE Information Security

Virtualized key protection and management for code signing, transaction signing, protecting secrets vaults, and more; virtualized encryption for storage and virtual machines; and key protection and support for any application using cryptographic services.

Note

This component was previously named **UKC** and may still be referred to it in some documents.

2. Unbound CORE Identity Security

Authenticate users and machines and protect PKI – seamlessly across all locations and devices. Create virtual enclaves for mobile and for desktop with maximum security and without sacrificing the user experience.

Note

This component was previously named **CoT** and may still be referred to it in some documents.

3. Unbound CORE Crypto Asset Security

Enterprise software for providing multi-party transaction approval and policy validation for banks, financial institutions, and enterprises that can accommodate any crypto asset or blockchain ledger.

Quickly adapt to new assets, services, and workflows as required by law or by market necessity with our crypto-asset security solution build for business. Accommodate any crypto asset or blockchain ledger with multiparty transaction approvals, financial institution policy validation, and defined risk-based policies.

Note

This component is referred to as **CASP** in some documents.

3. 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**.

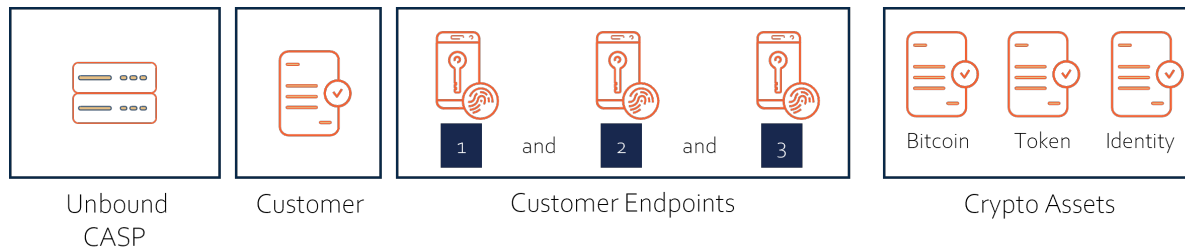


Figure 1: CORE Crypto Asset Solution Components

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.

3.1. 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](#).

3.1.1. 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.

3.1.2. 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.

3.1.3. 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.

3.1.4. 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.

3.1.5. 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.

3.1.6. 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.

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

3.1.7. 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.

3.1.8. 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.

3.2. 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.

3.3. Solution Architecture

The CASP architecture is shown in the following figure.

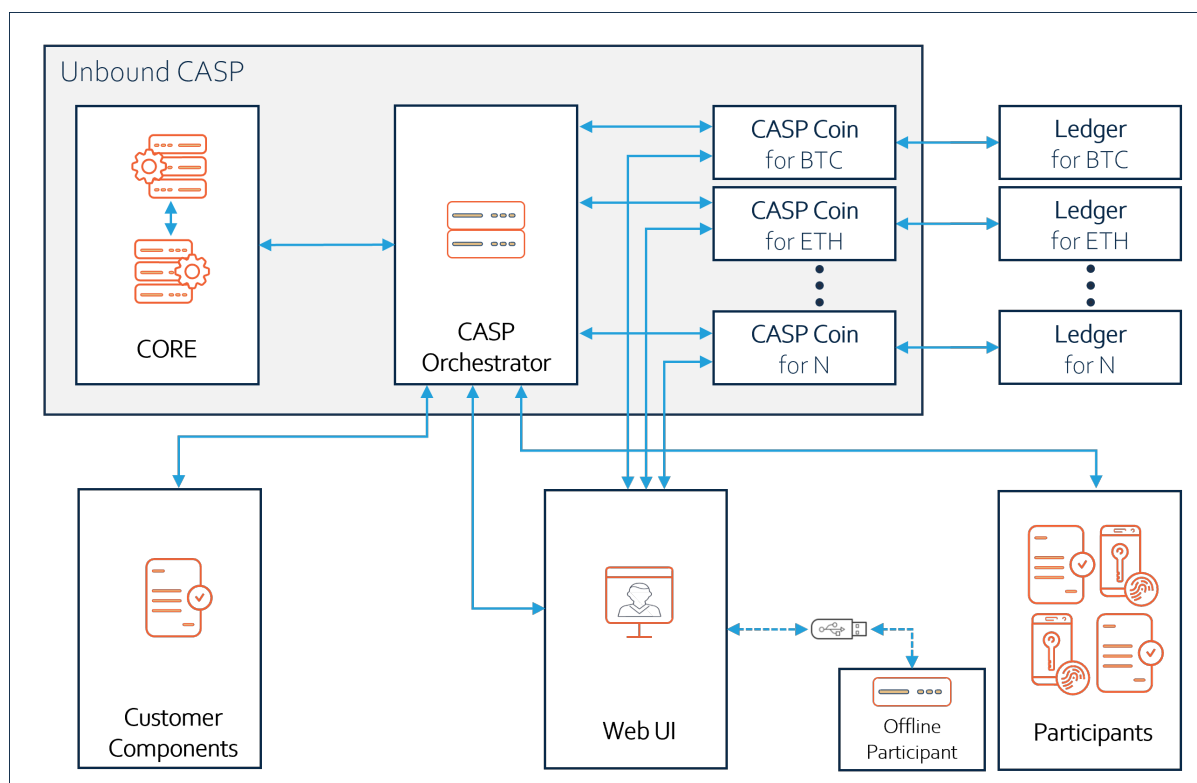


Figure 2: CASP Architecture

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.

4. System Requirements

Unbound CORE Crypto Asset Security ("CASP") software runs in the framework of Tomcat. It uses Java and makes use of JDBC (Java Database Connectivity) to connect to the database.

CASP requires the following:

4.1. Server Requirements

Each server in the CASP solution has the following requirements:

Hardware

- CPU: 2.0 GHz 64-bit (Intel, AMD, [IBM Z 15](#))
- RAM: 4 GB
- Disk space: 200 MB

OS

- Linux RHEL/CentOS 7.2
- Ubuntu 16.04 or 18.04
- [LinuxONE III](#)

Java

- Oracle Java JDK 8
- OpenJDK 11. OpenJDK can be obtained from several different sources. The recommended source is [AdoptOpenJDK](#) with the options for *OpenJDK 11* and *HotSpot*.

Database

- [PostgreSQL](#) 9.6 and 11
 - Compatible with [Amazon Aurora](#)
 - Compatible with [Amazon RDS](#)
- [MySQL](#) 5.7
 - Compatible with [Amazon Aurora](#)
 - Compatible with [Amazon RDS](#)
 - Compatible with [Azure MySQL](#) version 8

Unbound CORE Information Security (CORE)

- The CORE sub-version (2.0.x) must match or be newer than the CASP sub-version (1.0.y). For example, if you have CASP 1.0.1904, you must have at least CORE 2.0.1904.

Other Software

- NodeJS v10.13.0 or newer, but not including version 11
- Apache httpd 2.4.6

Communication Ports

The following ports are required to be open to allow communication between CASP components.

TCP Port	From	To	Comment
443	CASP Orchestrator	CORE Entry Point server	CASP/CORE communication
443	Web browser	CASP Orchestrator	UI Management
5432	CASP Orchestrator	CASP database	RDS PSQL or Aurora PSQL (if used)
3306	CASP Orchestrator	CASP database	RDS MySQL or Aurora MySQL (if used)
443	CASP Participant	CASP Orchestrator	CASP to Participants
443	BYOW ledger	CASP Orchestrator	Uses the CASP APIs
443	Customer application	CASP Orchestrator	Uses the CASP APIs

4.2. Client Requirements

For iOS:

- Hardware
 - iPhone 5s or later
 - iPad (5th generation and later)
 - iPad Pro
 - iPad Air 2
 - iPad mini 3 or later
- OS 10
- Face ID or Touch ID is required

Note

Registration with Face ID or Touch ID is required before activation with the CASP SDK. Changing or adding a fingerprint or face requires reactivation with CASP.

For Linux:

- Hardware
 - CPU: 2.0 GHz 64-bit (Intel, AMD)
 - RAM: 2 GB
 - Disk space: 200 MB
- OS
 - Linux RHEL/CentOS 7.2
 - Ubuntu 16.04
- Java
 - Oracle Java JDK 8
 - OpenJDK 11. OpenJDK can be obtained from several different sources. The recommended source is [AdoptOpenJDK](#) with the options for *OpenJDK 11* and *HotSpot*.

4.3. Supported Browsers

The CASP [Web Interface](#) is supported on the latest stable version of Google Chrome. Other browsers may work, but are not tested.

4.4. Supported 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 provides built-in support for the following wallets:

- btc - [bitcoin](#)
 - A Blockset access token is required for bitcoin. CASP uses this service to communicate with the ledger. See the [Blockset documentation](#) for more information.
- btctest - bitcoin [Testnet3](#)
- eth - [Ethereum](#)
 - An Infura token is required for Ethereum. See [Infura](#) for more information.
- ethtest - Ethereum [testnet](#)
 - CASP supports these test networks: ropsten, kovan, and rinkeby.
- ERC-20 - Ethereum blockchain tokens that are compliant with [ERC-20](#). A list of supported tokens can be found [here](#).
 - An Infura token is required for Ethereum. See [Infura](#) for more information.

If you need to support a different coin type, see [Bring Your Own Wallet \(BYOW\)](#).

4.5. System Limits

The CASP system has limitations on the maximum number of items that can be created, as detailed in the following table.

Item	Per account	Per vault	Globally
Accounts			1K
Active Participants	64		128
Active Data collectors	64		128
Users	64		128
Vaults			10k
Sub accounts		1k	10k
Keys			100k

These limits were verified to work by Unbound. If your organization needs a different limit [contact Unbound Support](#).

5. Installing CASP

This chapter walks you through the Unbound CASP service installation and activation process. If you are upgrading from a previous version of CASP, follow the procedure in [Upgrading CASP](#).

CASP installation includes the following steps:

- [Prerequisites](#)
 1. [Install Java](#)
 2. [Set up NodeJS](#)
 3. [Set up Apache HTTP Server](#)
 4. [Setup CORE](#)
 5. [OpenSSL Configuration](#)
- [CASP RPM Installation](#)
- [CASP Configuration](#)
 1. [Configure the CASP Database](#)
 2. [CASP Setup Utility](#)
 3. [CASP Configuration File](#)
 4. [Wallet Configuration](#)
 5. [Start the CASP Services](#)
 6. [Configure Apache](#)
 7. [Configure Logs](#)
 8. [Push Notifications \(optional\)](#)
- [Testing the CASP System](#)
- [Next Steps](#)

5.1. Prerequisites

Installation instructions for the prerequisites are provided in the following sections.

5.1.1. Install Java

1. Determine which version of java you have by running the following command:

```
java -version
```
2. If you are not running one of the versions of Java described in [System Requirements](#), download and install it.

5.1.2. Set up NodeJS

NodeJS is required for the built-in wallets. If you are using BYOW, then you can skip this step. To install NodeJS, follow the instructions found on the NodeJS website:

<https://nodejs.org>

For example, for CentOS or Ubuntu, following the instructions described in [NodeJS Package Managers](#).

To verify that NodeJS is installed, run the following command:

```
node -v
```

Tip

You may want to use a NodeJS package manager, such as [n](#) or [nvm](#), to help manage the NodeJS version.

5.1.3. Set up Apache HTTP Server

Install Apache HTTP daemon.

- For CentOS:

```
sudo yum -y install httpd
sudo yum -y install mod_ssl
```

- For Ubuntu:

```
sudo apt-get install -y apache2
sudo a2enmod ssl
sudo a2enmod proxy
sudo a2enmod proxy_http
```

5.1.4. Setup CORE

Setting up CORE, involves installing CORE, configuring it for CASP, and configuring the backup key.

1. Install CORE.
 - a. Install the CORE [entry point \(EP\) and partner servers](#).

Note

Only use the FIPS installation instructions if you specifically need it.

- b. For BIP implementations, [install an auxiliary server](#).
- c. Refer to the [CORE User Guide](#) for more information about this process.

2. Configure CORE for CASP.
 - a. [Create a new partition](#) named *casp* by specifying the `--default_client 1` option.

Note

1. To access this partition, the username is *user*.
2. If you have multiple CASP systems, each one needs a dedicated partition.
3. If you create a new user, it must have the *user* role, not the *so* role.

- b. [Reset the default user password](#) in the new CASP partition.
3. Create the cold backup key (which can be a 2048, 3072 or 4096 bit key). This key is used to encrypt the key material. The backup private key must be strongly segregated from the CASP system. The following are possible methods for creating this key.

- a. Use another CORE system, segregated from your CASP environment (that is, not the CORE that you are using for this CASP implementation).
- b. Use an offline device that is not connected to the network. You can run the following command on your offline device to create an RSA 2048 key pair.

```
openssl genrsa 2048 > key.pem
```

- c. Use an HSM.

Note

It is recommended to put the private key in cold, disconnected storage. The key can also be duplicated for resiliency.

4. Install the cold backup key.

A file with the public part of this key must be installed on the Entry Point (EP) and Partner of one server pair. It serves as the CORE backup encryption key for databases located on the EP and Partner servers.

- a. Create a file with the public key. For example, if your key is in a file called *key.pem*:

```
openssl rsa -in key.pem -pubout > casp_backup.pem
```

- b. Run the [CORE offline backup key](#) script, once on the EP server and once on the Partner server.

Note

If you have multiple CORE pairs, you only need to run the script on one pair.

5. You can verify the CORE installation by running a curl command. See [Checking CORE Service](#) for more information.

Note

To restore, follow the procedure described in [CASP Restore Utility](#).

5.1.5. OpenSSL Configuration

Check that your version of OpenSSL supports the required curves. List the curves using this command:

```
openssl ecparam -list_curves
```

The output should contain these curves:

```
secp256k1 : SECG curve over a 256 bit prime field
secp384r1 : NIST/SECG curve over a 384 bit prime field
secp521r1 : NIST/SECG curve over a 521 bit prime field
prime256v1: X9.62/SECG curve over a 256 bit prime field
```

If those curves are not found in the output, you need to update your version of OpenSSL. For example, on CentOS run:

```
sudo yum update openssl
```

5.2. CASP Package Installation

Download and install the **CASP Service**.

1. Access the CASP repository from the link provided to you by Unbound.
2. Select your platform and version and then download the corresponding package, *casp-{version}.rpm* or *casp-{version}.deb*.
3. (Optional) Verify the package as described in [Unbound Public Key](#).
4. Install the package file.
 - For CentOS:

```
sudo yum -y install casp-{version}.rpm
```

- For Ubuntu:

```
sudo dpkg -i casp-{version}.deb
```

5.2.1. CASP Configuration

The following steps describe configuration that is needed after installing the CASP package. The section [CASP Setup Utility](#) is not required but can be helpful with configuration. Follow the rest of the sections as needed.

Warning

CASP requires that you set up the CORE backup, as described in [Backup](#).

5.2.2. Configure the CASP Database

1. On the CASP Orchestrator, create the CASP database and a database user. See [Server Requirements](#) for a list of supported databases. The database user and password are needed for the *casp.conf* file as described in the next section.

- For MySQL:

```
mysql -u username -p -e "CREATE DATABASE casp CHARACTER SET utf8mb4  
COLLATE utf8mb4_unicode_ci;"
```

- For PostgreSQL:

```
psql -U username -c 'CREATE DATABASE casp;'
```

Or:

```
createdb casp
```

See [PostgreSQL](#) for more information on installing PostgreSQL.

2. Load the schema file from directory */opt/casp/sql*. Use the file that corresponds to your database.

- For MySQL:

```
mysql -u username -p database < /opt/casp/sql/casp-mysql.sql
```

For example, for a database named *casp*, with a user *sa*, and password *abc123*:

```
mysql -pabc123 casp -u sa
```

- For PostgreSQL:

```
psql -U username casp < /opt/casp/sql/casp-postgresql.sql
```

5.2.3. CASP Setup Utility

CASP contains scripts that can help with the installation. They are found in the *bin* directory and provide these functions:

- [CORE Password Setup](#) - encrypts the CORE password.
- [CORE Health Check](#) - returns the health of CORE.
- [Database Password Setup](#) - encrypts the database password.
- [Database Health Check](#) - returns the health of the database.
- [Firebase Token Setup](#) - configures the Firebase token used for push notifications on iOS/Android.

Each of these scripts and their associated options are explained in the following sections.

5.2.3.1. CORE Password Setup

This script encrypts the CORE password that is used in the [CASP Configuration File](#). The script uses the *casp.conf* file if it exists. If it does not exist, the script creates the file.

Syntax:

```
sudo casp_setup_ukc \  
--ukc-url=<ukc_url> \  
--ukc-user=<ukc_user> \  
--ukc-password='<ukc_password>' \  
--ukc-admin-user <ADMIN-USERNAME> \  
--ukc-admin-password <ADMIN-PASSWORD> \  
--force-get-ukc-ca
```

Note

This command must be called with `sudo`.

The parameters are described in the following table.

Parameter	Required	Description
ukc-url	required	URL of the CORE.
ukc-user	required	CORE user name. The format is the user name, an @ sign, and then the partition name. For example, "user@casp".
ukc-password	optional	CORE password. If it is not specified on the command line, the utility prompts for it. This password was set up in Setup CORE , step 2b. Use single quotes if the password contains special characters.
ukc-admin-user	optional	SO username used to configure Single Sign-On .
ukc-admin-password	optional	SO password used to configure Single Sign-On .
force-get-ukc-ca	optional	The first time that the utility runs it downloads the certificate (in PKCS #7 format) for the CORE server. Subsequent runs use the same certificate, unless this flag is specified.

Example:

```
sudo casp_setup_ukc --ukc-url <IP ADDRESS OF CORE EP> --ukc-user user@casp --ukc-  
password '<CORE PASSWORD>'
```

Warning

You must restart the CASP services for the changes to take effect, using the command:
`sudo service casp.tomcat restart`

5.2.3.2. CORE Health Check

This script checks the health of the CORE.

Syntax:

```
casp_check_ukc
```

There are no parameters for this script.

The response for a correctly working CORE is:

```
UKC health check was successful
```

5.2.3.3. Database Password Setup

This script encrypts the database password that is used in [Configure the CASP Database](#).

Syntax:

```
sudo casp_setup_db \
--db-url=<DB_url> \
--db-user=<DB_user> \
--db-password=<DB_password> \
--db-driver=<org.postgresql.Driver> \
--db-driver-path=<postgresql-9.6.jar>
```

Note

This command must be called with `sudo`.

The parameters are described in the following table.

Parameter	Required	Description
db-url	required	URL of the database.
db-user	required	Database user name.
db-driver	required	The database driver, such as <i>org.postgresql.Driver</i> or <i>com.mysql.jdbc.Driver</i> .
db-driver-path	required	Path to the database driver, such as <i>postgresql-9.6.jar</i> or <i>/opt/casp/jdbc/mysql-connector-java-8.0.12.jar</i> .
db-password	optional	Database password. If it is not specified on the command line, the utility prompts for it. Note: If the password is specified on the command line, it remains in the history of the shell where the command was run.

Example:

```
sudo casp_setup_db --db-url jdbc:postgresql://casp.abc123.us-east-1.rds.amazonaws.com:5432/casp --db-user myDBuser --db-driver org.postgresql.Driver --db-password abc123456 --db-driver-path /opt/casp/jdbc/postgresql-42.2.5.jar
```

Warning

You must restart the CASP services for the changes to take effect, using the command:
`sudo service casp.tomcat restart`

5.2.3.4. Database Health Check

This script checks the health of the database.

Syntax:

```
casp_check_db
```

There are no parameters for this script.

The response for a correctly working database is:

```
Database health check was successful
```

Tip

You can check the database health with the [Database Health Check](#) script, the [Status API](#), or in the [Web Interface](#).

5.2.3.5. Firebase Token Setup

This script configures the Firebase token that is used for push notifications on iOS/Android.

Syntax:

```
casp_setup_fb_token \  
-token <arg>
```

The parameters are described in the following table.

Parameter	Required	Description
token	required	Firebase token.

5.2.4. CASP Configuration File

You must verify all the settings in the `/etc/unbound/casp.conf` file. The default configuration file is created when CASP is installed. Replace all values in angle brackets ("`<>`") with the desired values. You must uncomment the lines related to the database that you are using (either MySQL or PostgreSQL). All other lines that are commented are optional.

Notes

1. If you ran the [CORE Password Setup](#) or [Database Password Setup](#) script, you see encrypted passwords in this file.
2. Passwords containing the following characters should not be used in the `casp.conf` file: `%`, `!`

```
# Database Settings (PostgreSQL) - Uncomment if using PostgreSQL  
#database.url=jdbc:postgresql://localhost:5432/casp  
#database.user=postgres  
#database.password=123456  
#database.driver=org.postgresql.Driver  
#database.driverfile=/opt/casp/jdbc/postgresql-<VERSION>.jar  
  
# Database Settings (MySQL) - Uncomment if using MySQL  
#database.url=jdbc:mysql://localhost:3306/casp?useUnicode=true&characterEncoding=  
UTF-8  
#database.user=root  
#database.password=12345678  
#database.driver=com.mysql.cj.jdbc.Driver  
#database.driverfile=/opt/casp/jdbc/mysql-connector-java-<VERSION>.jar  
  
# UKC Settings  
ukc.url=https://<UKC URL>
```

```
ukc.user=<UKC user>@<UKC partition>  
ukc.password=<UKC user password>
```

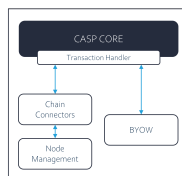
Tip

You can check the CORE health with the [CORE Health Check](#), [Status API](#), or in the [Web Interface](#).

5.2.5. Wallet Configuration

After installing CASP, you need to configure it to handle the crypto assets that you are using. CASP provides the infrastructure to handle various types of crypto assets, as shown in the following figure.

CASP Wallet Architecture



The heart of CASP is the CASP Core that handles all key management functionality. The [CASP Transaction Handler](#) is an interface that provides a way to customize the format of the transactions. Crypto assets are handled in one of these ways:

1. **Built-in wallets** - implemented using one of these methods for the chain connector:
 - a. Built-in support is provided for crypto asset ledgers for Bitcoin and Ethereum. The implementation is described below.
 - b. Plugins can be created to customize your chain connector for different types of crypto asset ledgers. Refer to the [Chain Connector](#) section of the *CASP Developers Guide* for information on how to create your own plugin.
2. **BYOW** - Bring Your Own Wallet - fully customized design using the Unbound APIs. See [here](#) for the full set of APIs provided to implement your wallet.

Configuration of the **built-in chain connectors** is described as follows.

Note

This configuration is needed only if you are using one of CASP's built-in crypto assets.

It is recommended to build your system using Bitcoin Testnet. Once the system is validated, switch to one that uses real currency.

Note

The log files for all wallets are in `/var/log/unbound/`.

Tip

You can check the chain connector health with the [Status API](#) or in the [Web Interface](#).

A script is provided that configures the CASP chain connectors. It can be used for

- [Bitcoin](#) (BTC)
- [Bitcoin Testnet](#) (BTCTEST)
- [Ethereum](#) (ETH)
- [Ethereum Testnet](#) (ETHTEST)

The script modifies the `/etc/unbound/wallets/development.yaml` file.

Note

All default values are taken from `/opt/casp/providers/wallets/config/default.yaml`. You must not edit this file since it is overwritten during an upgrade. However, you can use it for a reference for what options are available. To override a value, set it in `/etc/unbound/wallets/development.yaml`. For example, you can set `minTransactionValue` and/or `maxTransactionValue`. (The `min/maxTransactionValue` parameters are specified in the smallest unit for the asset, such as satoshi for Bitcoin and wei for Ethereum.)

Syntax:

```
sudo casp_setup_wallets \  
  --blockset-token <arg> \  
  --infura-project-id <arg> \  
  --log-level <arg>
```

The parameters are described in the following table.

Parameter	Required	Description
blockset-token	optional	Blockset API token.
infura-project-id	optional	Infura project ID for Ethereum.
log-level <arg>	optional	Change the default log level

For example, to configure BTCTEST:

```
sudo casp_setup_wallets \  
  --blockset-token abcd1234abcd1234abcd1234abcd1234 \  
  --infura-project-id dcba4321dcba4321dcba4321dcba4321
```

Response:

```
Wallets configuration was updated, wallets service must be restarted.
```

After running the script, restart the service with this command:

```
sudo service casp.wallets restart
```

You can then open the log file and make sure you see messages like these:

```
info: Using Blockset : BTC.main  
info: Using Blockset : BTC.test3
```

Troubleshooting

For both Bitcoin and Bitcoin Testnet, when retrieving the balance on a BIP44 vault with sub-accounts, you may encounter a situation where the Blockset BTC/BTCTEST token limit is reached. It returns an error code such as:

API calls limits have been reached.

The following parameters can be specified in the configuration file, *production.yaml*, to control the behavior of Blockset requests.

Blockset:

`maxConcurrentRequests: 10 // # of requests allowed to be sent at once to Blockset`

`minRequestTimeMs: 100 // The minimum time between one request and another`

`maxRetries: 10 // # of times to retry once a request to Blockset fails with 429 - too many requests`

5.2.5.1. Ethereum and Ethereum Testnet

- Installation of these wallets requires NodeJS.

Warning

You must set an Infura token for both ETH and EHTTEST.

- Optionally, update the fee settings. The file `/opt/casp/providers/wallets/config/default.yaml` contains the default fees. The file `/opt/casp/providers/wallets/config/development.yaml` lets you define your own fees.
- If needed, modify the type of test network, `Set network = "ropsten"`. It can have one of the following values: *ropsten*, *kovan*, or *rinkeby*
- Restart the service after any of these changes.

```
sudo service casp.wallets restart
```

5.2.5.2. ERC-20

ERC-20 is supported by the ETH/ETHTest wallet. To check an ERC-20 balance or create a transaction, add *contractAddress* to the request (refer to the [CASP API reference](#)).

A plain text name can be assigned to a contract address in the `/opt/casp/providers/wallets/config/development.yaml` file. The [Web Interface](#) uses these names when displaying transaction information. The configuration uses these fields:

- `address` - Token contract address.
- `name` - Name displayed to the user.
- `symbol` - Token currency symbol.
- `decimals` - Number of decimal digit precision supported by the token.

The following is an example showing the wallet setup for an Ethereum test network along with the ERC-20 token plain text names. It defines the name *My BOKKEY* for a contract address.

```

Wallets:
ETHTEST-ROPSTEN:
  walletType: ethereum
  name: Ethereum Ropsten
  signCompleteCallbackUrl: 'http://localhost:3000/ethtest-ropsten/api/v1.0/callback/withdrawals/complete'
  infura:
    network: ropsten
    token: 'abcd1234abcd1234abcd1234abcd1234'
  erc20Tokens:
    - address: "0xabcd1234abcd1234abcd1234abcd1234abcd1234"
      name: "My BOKKEY"
      symbol: "BOKKEY"
      decimals: 18

```

5.2.6. Start the CASP Services

This procedure configures the CASP services.

1. Start the service.

```
sudo systemctl start casp.tomcat
```

2. To check the status of a service on the system, use the `status` command:

```
sudo service casp.tomcat status
```

3. If you are using the built-in CASP wallet (BTC or ETH), start the service.

```
sudo systemctl start casp.wallets
```

5.2.7. Configure Apache

Note

You can access CASP over HTTPS using Apache. By default, there is an Apache self-signed certificate that can be used for testing. This certificate must be replaced for production. To replace the certificate, update the CASP configuration for Apache in `/etc/httpd/conf.d/casp-apache.conf`. See the [Apache documentation](#) for more information.

1. Configure Apache to start on boot.

- For CentOS:

```
sudo systemctl start httpd
sudo systemctl enable httpd
```

- For Ubuntu:

```
sudo systemctl start apache2
sudo systemctl enable apache2
```

2. To verify that Apache is running, navigate to `http://<server-ip>`. Replace `<server-ip>` with the IP address of the server. You should see an Apache landing page if it is running.

3. To configure Apache to only use HTTPS:
 - a. Edit `/etc/httpd/conf.d/casp-apache.conf`.
 - b. Comment out all the lines for the HTTP virtual host. This is the section of the configuration file that starts with `<VirtualHost *:80>` and ends with `</VirtualHost>`.
 - c. Edit `/etc/httpd/conf.d/httpd.conf`.

Note

The location and name of the `httpd.conf` file depends on the OS and version. For example, on Ubuntu, the file is `/etc/apache2/apache2.conf`.

- d. Comment out the `Listen 80` line in `httpd.conf`.
4. Restart Apache:

```
sudo systemctl restart httpd
```

5.2.8. Configure Logs

Use the following procedure to enable trace logging. Unbound support may ask you to create this log if you encounter an issue with CORE.

1. Edit `/etc/unbound/log4j/casp.xml`.
2. Find the following line:

```
<Logger name="TRACE" additivity="false" level="off">
```

3. Change it to:

```
<Logger name="TRACE" additivity="false" level="debug">
```

5.2.9. Push Notifications (optional)

The Unbound CASP Signer app (see [Mobile App](#)) uses Google *firebase* for push notifications on iOS and Android.

Note

Enabling push notifications is optional. The mobile app is fully functional without push notifications, but the user has to open the app to manually check if there are new approval requests.

A script is provided to setup the *firebase* token. This script checks the CORE health and then stores the *firebase* push token in CORE. The token is used to send push notifications to the [Mobile App](#).

Syntax:

```
sudo casp_setup_fb_token \  
--token=<TOKEN>
```

The parameters are described in the following table.

Parameter	Required	Description
token	required	The push notification token.

The response is:

```
UKC health check was successful.  
Firebase token set up successfully.
```

After running the script, restart the Tomcat service.

```
sudo service casp.tomcat restart
```

5.3. Testing the CASP System

The following commands can be used to test the components of CASP.

Note

There is a utility described in [CASP Setup Utility](#) that can be used to check the health of the CORE and the database.

■ Test CASP:

```
curl -k https://localhost/casp/api/v1.0/mng/status?withDetails=true
```

Response:

```
{ "version": "1.0", "ukc": { "healthy": true }, "db": { "healthy": true },  
  "nativeCrypto": { "healthy": true } }
```

■ Test CORE:

```
curl -k -X GET https://<Entry-Point-IP-address>/api/v1/info
```

Response:

```
{ "version": "2.0.1811.31664" }
```

■ Test BTC wallet:

```
curl -k https://localhost/btc/api/v1.0/status?withDetails=true
```

Response:

```
{ "version": "1.0", "ledger": { "healthy": true }, "casp": { "healthy": true  
  } }
```

■ Test BTCTEST wallet:

```
curl -k https://localhost/btctest/api/v1.0/status?withDetails=true
```

Response:

```
{ "version": "1.0", "ledger": { "healthy": true }, "casp": { "healthy": true  
  } }
```

■ Test ETH wallet:

```
curl -k https://localhost/eth/api/v1.0/status?withDetails=true
```


Response:

```
{ "version": "1.0", "ledger": { "healthy": true }, "casp": { "healthy": true } }
```

- Test ETHTEST wallet:

```
curl -k https://localhost/ethtest/api/v1.0/status?withDetails=true
```

Response:

```
{ "version": "1.0", "ledger": { "healthy": true }, "casp": { "healthy": true } }
```

5.4. Next Steps

To use CASP for testing with minimal integration effort, use the CASP built-in [Web Interface](#) and [Mobile App](#) for iOS.

To start integrating CASP with your backend systems and your different clients, see the [CASP Developers Guide](#).

You may want to implement a [Reverse Proxy](#) for all the endpoints that use the REST API. The reverse proxy redirects API calls going from the server to the CASP participant REST API endpoints.

6. Upgrading CASP

If you are upgrading CASP from a previous release, follow this procedure.

Note

After upgrading CORE and the schema, you cannot create new entities (e.g. vaults, participants) until CASP is updated.

1. **Upgrade CORE** - Instructions can be found in the [CORE User Guide](#).
 2. **Upgrade the schema** - The schema package can be downloaded from the Unbound repository. Uncompress the package and then load the schema file that corresponds to your upgrade:
 - For MySQL:

```
mysql -u [username] -p [database] < casp-mysql-upgrade.sql
```
 - For PostgreSQL:

```
psql [database] < casp-postgresql-upgrade.sql
```
 3. **Upgrade the bot** (if you are using a bot) - Use the instructions in [CASP Bot](#). If you are upgrading the CASP bot and using a directory different from the previous installation, you must copy the key file(s) from the old installation. Key files are found in: `<bot_directory>/bin/<key>.p12`
 4. **Install the new package** - Use the instructions in [CASP Package Installation](#).
 - For CentOS installations, check the configuration files. During the RPM installation, if there are configuration files that have manual edits, they are **not** overwritten. Instead, a new file is created with a *rpmnew* suffix. Check this file and copy any new, relevant configuration information to the original file.
- ### Note

During an upgrade, the following warnings can be safely ignored:

```
rm: cannot remove '//opt/casp/providers/bch/casp-bch-provider.jar': No such file  
rmdir: failed to remove '//opt/casp/providers/bch': Directory not empty
```
- For Ubuntu installations, the installer automatically asks what you want to do with files that already exist.
5. **Built-in Bitcoin configuration** - If you are using the built-in Bitcoin wallet and made changes to the default *casp-apache.conf* file, after upgrading you need to check the Bitcoin ports in the configuration files.
 - a. Edit the file */etc/httpd/conf.d/casp-apache.conf*.
 - b. Change the ports used for Bitcoin to port 3000 instead of 8080.
 - c. Restart the Apache service.
 6. **CASP wallets** - To upgrade CASP wallets, follow the instructions found [here](#).

7. **Web UI** - If you are using the web interface, you must clear the cache in your browser.
8. **Test the installation** - Use the instructions in [Testing the CASP System](#).

Note


After upgrading the CASP package, check the following to ensure that CASP points to the correct SQL driver and check that the driver exists.

1. Check for the driver in `/opt/casp/jdbc`. For example, for PostgreSQL, a file called `postgresql-<version>.jar` should exist.
2. In the `casp.conf` file, check the driver file setting. For example, for PostgreSQL, the `casp.conf` should contain:

```
database.driverfile=/opt/casp/jdbc/postgresql-<version>.jar
```

7. Single Sign-On

CASP can leverage CORE integration with an OpenID Connect (OIDC) provider enabling the Single Sign-On (SSO) authentication for the CASP partition users. After configuring CASP for OIDC, the CASP UI login page presents the standard login option and the registered SSO provider.



unbound
security

Login to your account [?](#)

Username

Password

☐ Remember my login for this session

Continue with CASP

OR

Continue with Google

Continue with Ping

To configure SSO, use the following steps.

1. Configure SSO in CORE by following the steps in the [CORE User Guide section on Single Sign-on](#).

Note

1. CASP uses the **email** (if provided) and **login name** for the OIDC aliases.
2. The identity provider used by CASP is currently limited to the **first** provider defined in CORE if more than one is defined.

2. Add the credentials of a CORE SO user to CASP. The SO user can be from any CORE partition. CASP uses this SO to create users in CORE that map to OIDC users. Set up the CORE username and password in CASP with the [CASP Setup Utility](#).

```
casp_setup_ukc --ukc-admin-user <SO-USER> --ukc-admin-password <SO-PASSWORD>
```

Note

If you leave out the *ukc-admin-password* argument, you are prompted to enter the password.

3. If Tomcat is not running, start it using the command:

```
sudo service casp.tomcat start
```

If Tomcat is running, restart it using the command:

```
sudo service casp.tomcat restart
```

4. In the CASP UI, add an OIDC user.
 - a. Access the **Users** screen.
 - b. Click **Create**.
 - c. Select *OpenID Connect* for the **Authentication method**.
 - d. Enter the email and the role.

New user

Name

John Smith

Authentication method

OpenID Connect

Login name

john.smith

Email

john@smith.com

Role

Super user

Cancel

Create user

There is now an OIDC user defined in CASP. That user can access the CASP UI with their OIDC credentials.

8. Reverse Proxy

It is a best practice **not** to have the CASP REST API completely open to the internet. However, you may want to enable internet access to specific REST API endpoints, such as the mobile devices that use your CASP app.

To accomplish this task, it is possible to implement a reverse proxy for all the endpoints that use the REST API. The reverse proxy redirects API calls going from the CASP participant REST API endpoints to the server.

Use the following procedure to create the reverse proxy. This example shows which API's should be open to make the system operational.

1. Install Apache (or any other web server) on the device that you are using for the proxy.
2. Create a file *casp-apache.conf* in the directory */etc/httpd/conf.d*.
3. The file should have the following contents.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule headers_module modules/mod_headers.so
Header set X-Content-Type-Options nosniff
Header set Content-Security-Policy "default-src 'self' 'unsafe-inline'"

<VirtualHost *:80>
    ProxyPreserveHost On
    ProxyPass /casp/api/v1.0/ep http://<CASP Server>/casp/api/v1.0/ep
    ProxyPassReverse /casp/api/v1.0/ep http://<CASP Server>/casp/api/v1.0/ep
</VirtualHost>

<VirtualHost *:443>
    ProxyPreserveHost On
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/localhost.crt
    SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
    ProxyPass /casp/api/v1.0/ep http://<CASP Server>/casp/api/v1.0/ep
    ProxyPassReverse /casp/api/v1.0/ep http://<CASP Server>/casp/api/v1.0/ep
</VirtualHost>
```

Note

The default certificate that is installed by *httpd* is a self-signed certificate. This should be replaced with your production certificate.

4. Restart *httpd*.

```
sudo service httpd restart
```

5. In the CASP app, use the URL of the proxy instead the server.

After entering the URL of the proxy in the CASP app, if it connects, then the proxy is configured correctly.

9. Key Storage

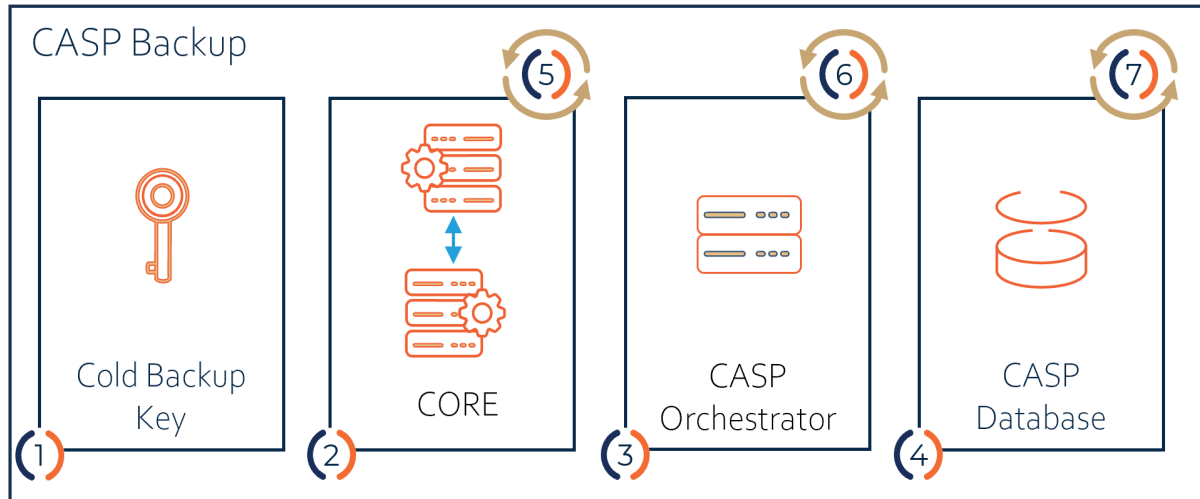
Keys are stored in vaults. A vault is a secure container for the cryptographic material used to protect a crypto asset, such as the seed or private key. CASP uses Multiparty Computation (MPC) to split the crypto material between the different participants in the vault, which ensures that the material 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.

Use the relevant endpoint based on the currency you want to work with. The list of supported currencies and the matching short names can be found in [Built-in Wallets](#).

10. Key Backup and Restore

CASP key material is shared across the CORE and the participants, and CASP management data is stored in the CASP DB. Backup of this data is divided into steps that are done during setup and steps needed in an ongoing basis. The backup data can be used to recover from a variety of different scenarios.



The numbers in the figure correspond to the respective steps in the next section.

10.1. Backup

Setup Tasks for Backup

During setup, perform these tasks:

1. **Create cold backup key for CORE.**

The cold backup key can be used to restore CASP from a loss of the entire system. It is set up during CASP installation (see [Setup CORE](#)).

2. **Setup the CASP backup.**

CASP has built-in support for secure backup and restore. Use the CORE procedures, which can be found in the [CORE User Guide](#).

Note

If you are using multiple CORE pairs, you only need to back up one pair.

3. Create a CASP quorum with redundancy.

Your quorum groups should have more participants than the required minimum number for approval. This setup allows replacing a participant, which may be necessary if someone loses their phone or someone leaves the company. For example, your group has 3 participants required for approvals, and it has 4 participants. A participant loses his phone, and the remaining 3 can reactivate the participant on a new phone.

Consider creating a couple of bots that are used as participants. These bots can be safely stored and only used when needed to meet the minimum quorum for approval.

4. Setup the CASP database backup.

Use the methods described by each database vendor for backup of your CASP DB:

- [MySQL backup and restore](#)
- [PostgreSQL backup and restore](#)

It can be helpful to save a copy of your *casp.conf* configuration file.

Ongoing Backup Tasks

The following are ongoing backup tasks to execute periodically:

5. Backup the CORE.

Backup CORE periodically. Use the CORE procedures, which can be found in the [CORE User Guide](#).

6. Backup the CASP vaults.

Backup a CASP vault immediately after it is created. Use the APIs found in the [CASP API descriptions](#).

- For BIP32 vaults, it backs up the first key that was derived normally on the derivation chain. For example:
 - For the chain `m/0'/1'/2/3/4` - it backs up only the 2 key.
 - For the chain `m/1/2/3/4` - it backs up only the 1 key.
- For BIP44 vaults, it backs up only the account key.

7. Backup the CASP database.

Use the methods described by each database vendor for periodic backups of your CASP DB:

- [MySQL backup and restore](#)
- [PostgreSQL backup and restore](#)

Note

Store the cold backup key and vault backup outside of the CASP/CORE servers, preferably off-site, to be used in the case of a complete system failure. Follow your best practices for storage of other backup data.

10.2. Recovery Scenarios

The following sections present different recovery scenarios and what is required to restore functionality.

CASP provides the [Status API](#) that returns the health of the components. You can run this command to help determine which component has an issue. In addition, you can test the components with the commands in [Testing the CASP System](#).

10.2.1. Lost Phone, Damaged Bot, or Employee Replacement

There are several similar situations that have similar recovery procedures:

1. **Lost phone** - one of the participants in your quorum group loses their phone. This participant gets a new phone and needs to be reactivated by the remaining participants.
2. **Employee replacement** or a **damaged bot** - an employee who is a participant in your quorum group either switches positions or leaves the company. Alternatively, you have a bot that gets damaged. This participant now needs to be replaced in the quorum.

The CASP vault implementation has built-in support for these scenarios. For a lost phone, the participant can just be reactivated. To replace a participant, deactivate the current participant from the relevant vault and then add the replacement participant to the vault.

These actions are part of the regular operations supported by CASP vaults and do not require any specific disaster recovery protocol. For details about adding participants to an existing vault, deactivating participants, and reactivating participants, refer to the [Participant Flows](#).

10.2.2. CORE Loss

Run the [Status API](#) to determine if there is a problem with the CORE. This API returns the health of the CASP components.

If a CORE server or cluster is lost or destroyed, you can bring up a new set of servers and then restore from your backups. See the [CORE Restore Procedure](#).

After restoring, rerun the [Status API](#) to determine all components are healthy.

10.2.3. CASP Server Loss

If the CASP server (i.e., the CASP Orchestrator) holding your CASP vaults is lost, you can install new CASP server. If you saved the *casp.conf* configuration file, copy it to the new server. If not, recreate the *casp.conf* file using the parameters for the existing CORE and database.

After restoring, rerun the [Status API](#) to determine all components are healthy.

10.2.4. CASP Database Loss

Run the [Status API](#) to determine if there is a problem with the database. This API returns the health of the CASP components.

If the CASP database is lost, you can use the methods described by each database vendor to restore the CASP backup:

- [MySQL backup and restore](#)
- [PostgreSQL backup and restore](#)

After restoring, rerun the [Status API](#) to determine all components are healthy.

10.2.4.1. Total System Loss

In case of a total loss of the CASP system, including CASP, CORE, and the database, the private keys can still be recovered if you have the cold backup key and the vault backup.

The previous methods are for restoring CASP components for recovery of the CASP system itself. However, there are cases where it is required to restore the actual vaults outside of at the CASP system, such as in a standalone compatible wallet.

Other use cases include:

1. An exchange system where the customer wants to be able to recover funds even if the exchange service ceases to operate. For this case, it can store the backup data in a different location and the backup key in a safe location outside of the exchange system.
2. Moving from CASP to a different vault system.

Using this approach requires the following steps:

1. Set up the backup key during CASP installation (see [Setup CORE](#)).
2. Get the backup data per [generated vault and sub-account](#).
3. Verify the backup information with the [CASP Restore Utility](#). This option allows you to verify backup data integrity without decrypting the key.
4. Use the backup data along with the private key to retrieve the original seed/key with the [CASP Restore Utility](#).

10.3. CASP Restore Utility

Unbound CASP includes a utility, called **CASP Restore**, to help with the backup and restore process. This utility is installed by the CASP RPM (and is also available as a separate download).

CASP Restore runs on these platforms:

- RHEL/CentOS 7.2 and later
- Ubuntu 16.04

Before using the utility, you need to have a backup.

1. To retrieve the backup data, use one of the following API requests:

- For a deterministic (non-hierarchical) vault use:

```
https://<casp-server>/api/v1.0/mng/vaults/{id}/backup
```

See [Get vault backup data](#).

- For BIP32 use:

```
https://<casp-server>/api/v1.0/mng/vaults/{vaultId}/backup/{publicKey}
```

See [Get backup data for public key](#).

- For BIP44 use:

```
https://<casp-server>/api/v1.0/mng/vaults/{id}/coins/{coin-number}/accounts/{account-number}/backup
```

Replace `{coin-number}` with the number for the coin type that you are using. You can see a list [here](#).

Replace `{account-number}` with your account number. You need to call this command once for each account number to retrieve a backup for each account.

Note that this is the backup of the account root key, not the actual key that holds the currency. All other account addresses can be derived from this key.

See [Get backup for account](#).

2. Save the returned JSON to a file for use in the verify/restore process.

Now that you have the backup you can use *casp-restore* to verify or restore the backup. The utility is called with the following command line:

```
casp_restore [options]
```

The utility has these options:

- `-op <restore or verify>` - Specify the operation that you need: *restore* or *verify*.
- `-alg <ecdsa or eddsa>` - Specify the algorithm: *eddsa* or *ecdsa*.
- `-in <file>` - the name of the backup file that includes the backup data.
- `-rsa <pem-file>` - the RSA *pem* file that corresponds to the CASP backup key. Use the private key for restore or the public key for verify.

Warning

You must use the key that was in use when the vault was created for vault restore. If you changed the backup key, the backup still requires the key that was used during vault creation for restore.

- `-rsa-pwd <password>` - the password for the RSA *pem* file.
- `-public-key <file>` - Specify the EC public key. Paste the string from the saved JSON file that was saved in the backup process.
- `-out <output pem file name>` - For a restore operation, the file that will hold the restored EC key.

- `-out-pwd <output pem password>` - For a restore operation, the password for the *pem* file.

10.3.1. Unbound Restore and Verify Utility

A utility is provided that can be used to verify the integrity of backup files. The utility is included in the CASP Restore package as source code. See the README file included in the package for details of how to install it.

10.4. Master Key Extractor

Note

This script only works for Bitcoin. For other wallets, please [contact Unbound Support](#).

A Python script, called **Master Key Extractor**, is provided along with the CASP Restore utility. Master Key Extractor creates a BIP32 master key from a *.pem* file. The *.pem* file is the output of the [CASP Restore Utility](#).

The script is called with the following command line:

```
masterkey_extractor.py [options]
```

The script has these options:

- `-p` (or `--pem`) - Private key *.pem* file. The file must not be in PKCS #8 format.
- `-l` (or `--level`) - Derivation level.
- `-c` (or `--cpar`) - cpar, chain code of the parent key.
- `-n` (or `--childnumber`) - Child number.
- `-f` (or `--parentfingerprint`) - Parent fingerprint.
- `-t` (or `--testnet`) - For *testnet* instead of *mainnet*.

The output of the script is a master key (in xprv format), like so:

```
abcd1234ABCD5678Z1CeAh2pKFwY8JLjVQDXUoFQwMbAuUgYctFnyA9jwFixgUJRRBVSVQAGK4VcXBHjB  
sWhqGhiuLQXj9habcd1234ABCD5678
```

The string includes the private key and other derivation information that was provided to the script. It can be used by other wallets to derive the keys that have actual funds associated with them.

10.5. Restore Examples

10.5.1. Restore a Deterministic (non-BIP) Vault

The following steps show an example of restoring a deterministic vault.

1. For this example, we use these parameters:
 - CASP server URL: 192.168.0.149
 - Vault ID: 381ab347-0e18-4bdf-8a7f-7f02332e655f
 - Name of the file containing the private key: *casp_backup_private.pem*
 - Coin type: Bitcoin Testnet
2. Get the backup data.

```
curl --request GET \
--url http://192.168.0.149/casp/api/v1.0/mng/vaults/381ab347-0e18-4bdf-8a7f-7f02332e655f/backup \
--header 'authorization: Bearer b3JpOjJkOTMyNGYzLTm0MjItNDgwNC1hYTkyLWY5ZThlYWExNmRmYWw=='
```

Response:

```
{
  "vaultID": "381ab347-0e18-4bdf-8a7f-7f02332e655f",
  "backupData": "ZWNiYWNRMDEAgALKAQAQ"
}
```

Note

The *backupData* field has been truncated for example purposes.

3. Save the contents (without the quotes) of the received *backupData* field to a file, called *backupData.txt*.
4. Get the public key.

```
curl --request GET \
--url http://192.168.0.149/casp/api/v1.0/mng/vaults/381ab347-0e18-4bdf-8a7f-7f02332e655f/publickey \
--header 'authorization: Bearer b3JpOjJkOTMyNGYzLTm0MjItNDgwNC1hYTkyLWY5ZThlYWExNmRmYWw=='
```

Response:

```
{
  "publicKey":
  "3056301006072A8648CE3D020106052B8104000A03420004D40671DD6DB05B96721BF854EF
  56E7EDD9FC10B4CED0090F365740513CFDE7A8691C6CA4ACBE307B0BCFBFF6F56397D2C015A
  D6C437CB2482DFA019DF23657D6"
}
```

5. Save the contents (without the quotes) of the received *publicKey* field to a file, called *publicKey.txt*.
6. Run the [CASP Restore Utility](#).

```
/opt/casp/bin/casp_restore -op restore -alg ecdsa -in backupData.txt -rsa
casp_backup_private.pem -public-key publicKey.txt -out ec-key
```

7. Convert the received key into plain text.

```
openssl ec -in ec-key -outform PEM -out restored.plain.pem
```

8. Run the [Master Key Extractor](#).

```
python /tmp/masterkey_extractor.py -p restored.plain.pem --testnet
```

You now have the master key (in *WIF* format), which is a string that includes the private key that was provided to the script. It can be used by other wallets that have actual funds associated with them.

To restore your wallet, use the master key in [Electrum](#):

1. In Electrum, select **File > New/Restore**.
2. Give the wallet a name and then click **Next**.
3. Select *Import Bitcoin addresses or private keys*, and then click **Next**.
4. Enter the master key, and then click **Next**.

A wallet is created using the master key.

10.5.2. Restore an HD (BIP44) Vault

The following steps show an example of restoring an HD vault.

1. For this example, we use these parameters:
 - CASP server URL: 192.168.0.166
 - Vault ID: db15bb8d-baec-4ceb-858c-7ae967261933
 - Name of the file containing the private key: *casp_backup_private.pem*
 - Sub-account number: 0
 - Coin type: Bitcoin Testnet
2. Get the backup data.

```
curl --request GET \
--url http://192.168.0.166/casp/api/v1.0/mng/vaults/db15bb8d-baec-4ceb-858c-
7ae967261933/coins/1/accounts/0/backup
```

Response:

```
{
  "backupData": "ZWNiYWNRmDEAgALKAQIAAAACo+aT5hBI2WZiNvmLR8dr4pe/MJ",
  "bipLevel": 3,
  "cpar": "772FB73BF97DF1E7E1A8BA6BC3F9656D8A633DF9873C0C2DBAC2003C65A6A652",
  "parentFingerprint": "2039A9A5",
  "childNumber": 0,
  "publicKey":
  "3056301006072A8648CE3D020106052B8104000A03420004E7A1172F667C3923C6C40ACA03
  E392457820B0F7CE144007919F4363BCF978EAE51188C10287A8E6FE211CFBB7166B0947AAB
  56324635AC60C0009296F28EC64"
}
```


Note

The *backupData* field has been truncated for example purposes.

3. Save the contents (without the quotes) of the received *backupData* field to a file, called *backupData.txt*.
4. Save the contents (without the quotes) of the received *publicKey* field to a file, called *publicKey.txt*.
5. Run the [CASP Restore Utility](#).

```
/opt/casp/bin/casp_restore -op restore -alg ecdsa -in backupData.txt -rsa  
casp_backup_private.pem -public-key publicKey.txt -out ec-key
```

6. Convert the received key into plain text.

```
openssl ec -in ec-key -outform PEM -out restored.plain.pem
```

7. Run the [Master Key Extractor](#). The parameters are from the response of the *backup* curl command.

```
python /tmp/masterkey_extractor.py -p restored.plain.pem \  
-l 3 \  
-c 772FB73BF97DF1E7E1A8BA6BC3F9656D8A633DF9873C0C2DBAC2003C65A6A652 \  
-n 0 \  
-f 2039A9A5 \  
--testnet
```

You now have the master key (in *xprv/tpv* format), which is a string that includes the private key and other derivation information that was provided to the script. It can be used by other wallets to derive the keys that have actual funds associated with them.

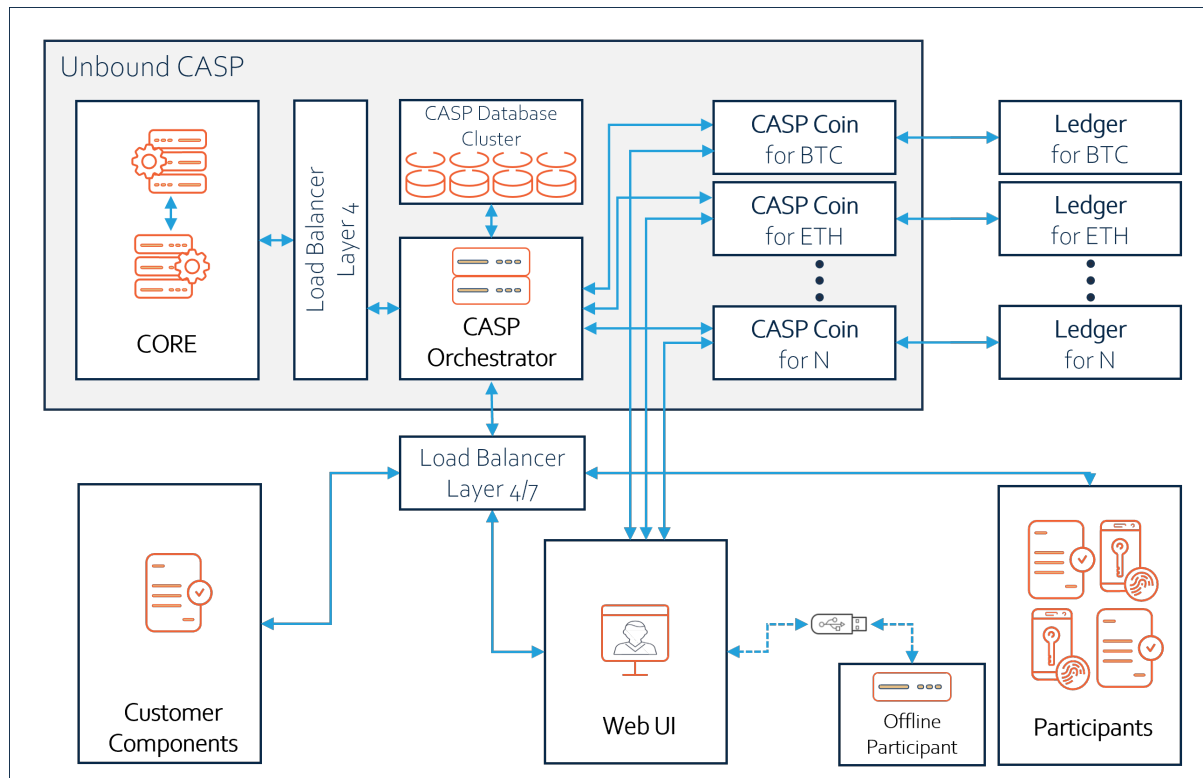
To restore your wallet, use the master key in [Electrum](#):

1. In Electrum, select **File > New/Restore**.
2. Give the wallet a name and then click **Next**.
3. Select *Standard wallet*, and then click **Next**.
4. Select *Use a master key*, and then click **Next**.
5. Enter the master key, and then click **Next**.

A BIP wallet is created using the master key.

11. High Availability

High availability ("HA") and scalability with CASP is accomplished with configuration of CASP components. The overall architecture for the highly available CASP system is shown in the following figure.



Two load balancers ("LB") are needed for the HA implementation. The first LB is between the CASP Orchestrator and the CORE, and operates on layer 4. The second LB is between the CASP Orchestrator and the participants, web clients, and other customer components, and operates on layer 4 or layer 7.

Configuration of the CASP components for high availability is described in the following sections.

11.1. CORE

The CORE system has built-in support for high availability through installing multiple CORE pairs. A production setup would therefore usually include 4 CORE servers (2 Entry Point and 2 Partner servers).

For details about building a high availability CORE system, refer to the CORE User Guide section on [Cluster Scale-Out](#).

Since CASP uses CORE with a REST API, you need to configure a load balancer for the different CORE entry points so that there is a single URL accessed by CASP.

To configure the load balancer:

Use the [ekm_renew_server_certificate](#) command to add the name of the load balancer to every EP, using the `--names` flag. The name should be the name defined in the CORE.

11.2. CASP Database

For your production environment you may want to require the CASP database (see [Configure the CASP Database](#) to have fail over support. Refer to the following links that detail how to add fail over support depending on your database type:

- [MySQL high availability](#)
- [PostgreSQL high availability](#)

11.3. CASP Orchestrator

After configuring the CORE and database to support HA, you can replicate the CASP Orchestrator server. This replication is accomplished by installing additional servers. Configure the replicated servers with the same *casp.conf* used for the first CASP Orchestrator. These servers can also be cloned with any standard auto load balancing services that are supported by the different cloud providers.

To allow your application to access different CASP Orchestrator servers, put a load balancer in front of all of the CASP Orchestrator servers, so they can be accessed with a single URL from your application servers.

11.4. Endpoints

Supporting HA for the CASP endpoints is inherent in the quorum structure of the CASP vaults. You can add redundancy to your vaults by adding additional participants and requiring a subset of them for a successful operation. See the [CASP Developers Guide](#) for details on how to create and use vaults with inherent redundancy.

12. Audit and Logging

CASP and CORE have log files to help monitor the system status and operations.

12.1. CASP Log Files

CASP has two log files, *casp.log* and *casp-trace.log*.

casp.log

- Path: */var/log/unbound/casp.log*
- Purpose: This log file can be used for auditing. It contains every CASP event, along with the associated initiator, operation, and duration.
- Structure: Each line in the log file has these fields:
 1. Date
 2. Time
 3. Threshold
 4. Event
 5. Owner
 6. Client
 7. Account ID
 8. Vault ID
 9. Participant ID
 10. Operation ID
 11. Operation Time
 12. Operation Message
 13. Error Details (optional)

Note

All fields can have the value *N/A*, except for the last field (Error Details), which can be blank.

For example:

```
2019-07-09 05:47:28,613 INFO POLICY_SATISFIED N/A N/A ba783c93-d79c-4406-
b5cb-c5fdd5e56e69 6b60cf0f-aa68-4b19-b46d-a38614884cab N/A N/A N/A 'Policy
P1 was satisfied for transaction: amount: 0.15 currency: ETH, destinations:
[001CF7DA364E337B0FCA476E725B7C099F3F8EA0]'
```

casp-trace.log

- Path: */var/log/unbound/casp-trace.log*
- Purpose: This log file is used for debugging. It contains very detailed information about all CASP activities. If you [contact Unbound Support](#) you may be asked to send this file.

Note

This file is only created if you enable trace logging as described in [Configure Log4j](#).

12.2. CASP Built-in Wallet Logs

The CASP built-in wallets have their own associated log files. These log files use the [winston](#) framework for logging.

Three log transports are supported, which can be configured in *config/production.yaml*:

- **DailyRotateFile** - daily rotated file. Turned on by default. The log file is stored in */var/log/unbound/wallets*. The log file is configured in *config/production.yaml*, which has these parameters (along with sample data):
 - level: info
 - dirname: '/var/log/unbound/wallets'
 - filename: 'wallets-%DATE%.log'
 - datePattern: 'YYYY-MM-DD-HH'
 - zippedArchive: true
 - maxSize: '20m'
 - maxFiles: '14d'
 - options:
 - mode: o640

Descriptions of the configuration parameters can be found in the [DailyRotateFile documentation](#).

Note

Over time, you may have many log files in the output directory. To view the latest one, you can use a command such as:

```
tail -f $(ls -t *.log | head -1)
```

- **console** - turned off by default.
- **file** - single log file. Turned off by default.

12.3. Configure Log4j

If you want to enable trace logging:

1. Edit */etc/unbound/log4j/casp.xml*.
2. Find the following line:

```
<Logger name="TRACE" additivity="false" level="off">
```

3. Change it to:

```
<Logger name="TRACE" additivity="false" level="debug">
```

12.4. Connecting CASP to Syslog

CASP logs can be forwarded to a specified syslog server for further analysis.

To forward CASP logs to a syslog server, perform the following steps on the CASP server:

1. Obtain the following info regarding the syslog server:
 - The hostname/IP address of the syslog server.
 - The type of IP protocol used to deliver the messages (UDP/TCP).
 - The *syslog* service port number.

Troubleshooting

To check that the connection is not blocked, telnet to the designated destination/port. For example, `telnet syslog.server.ip 514`.

2. Locate and open the `log4j.xml` file (refer to [Configure Log4j](#)) and add the following line before the `</Appenders>` tag:

```
<Syslog name="log" host="syslog.server.ip" protocol="TCP" port="514"
newLine="true"/>
```

The following is an example of the complete file `log4j` file:

```
<?xml version="1.0" encoding="UTF-8" ?>

<Configuration status="warn" name="CASP" packages="">
  <Properties>
    <Property name="baseDir">/var/log/unbound</Property>
  </Properties>
  <Appenders>
    <RollingFile name="log" fileName="${baseDir}/casp.log"
      filePattern="${baseDir}/${date:yyyy-MM}/casp.%d{yyyy-MM-dd}.log.gz">
      <PatternLayout pattern="%d %-5level %m %throwable %n"/>
      <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
      <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
          <IfFileName glob="*/casp.*.log.gz"/>
          <IfLastModified age="60d"/>
        </Delete>
      </DefaultRolloverStrategy>
    </RollingFile>

    <Syslog name="log" host="syslog.server.ip" protocol="TCP" port="514"
newLine="true"/>
  </Appenders>

  <Loggers>
    <Logger name="AUDIT" additivity="false" level="info">
      <AppenderRef ref="log"/>
    </Logger>

  </Loggers>
</Configuration>
```

Note

The *name* field must match the corresponding *ref* name, such as: `<AppenderRef ref="log">`

3. Replace the `syslog.server.ip` string with the syslog server IP address or hostname.
4. As needed, change the default protocol (TCP) and the default port (514).
5. Restart CASP, which is with this command:

```
sudo service casp.tomcat restart
```

6. Check that the syslog server captures the log message generated during the CASP restart.

Tip

By default, a syslog server uses the `/var/log/messages` file to store logs.

12.5. CORE Log Files

Descriptions of the log files for CORE can be found in the [CORE Maintenance Guide](#).

12.6. Troubleshooting Audit Logs

12.6.1. 500 Error

If you try and verify an audit log and receive a 500 error, it may be due to a problem with a table in the database. Audit verification can be done through these methods:

- [All audit data](#)
- [Audit verification by type](#)
- With the *fullValidation* flag. For example, see the [accounts audit](#).

The response is similar to the following error:

```
{
  "type": "/mng/errors/data-verification-failed",
  "title": "Validation of audit data failed",
  "details": "Row 24 verification failed",
  "status": 500
}
```

This issue can be fixed, either for all audit logs or for a specific type of audit log.

Note

It is recommended to backup the database before running the following procedure.

To reset all audit data:

1. Remove all audit data by running these commands on the database.

```
delete from casp_audit_signs;
delete from casp_audit_pools;
delete from casp_audit_accounts;
delete from casp_audit_trusted_systems;
delete from casp_audit_users;
delete from casp_audit_vaults;
delete from casp_audit_data_collectors;
```

2. Run *casp_delete_ukc_secrets* utility, found in the *casp-service/bin* directory. On success, you see:

```
Deleting CASP audit secrets from UKC succeeded
```

3. Restart the CASP server.

To reset audit data for a specific type of log, such as *accounts*:

1. Remove the audit data by running the command below on the database, corresponding to the type of log.

```
delete from casp_audit_signs;
delete from casp_audit_pools;
delete from casp_audit_accounts;
```



```
delete from casp_audit_trusted_systems;  
delete from casp_audit_users;  
delete from casp_audit_vaults;  
delete from casp_audit_data_collectors;
```

2. Log into the CORE user interface.
3. Access the **Secrets** screen.
4. Delete the two secrets (*first-parent-hash* and *last-hash*) corresponding to the type of log that you are resetting. Secret names are in this format:

```
casp-audit-<LOG TYPE>-first-parent-hash  
casp-audit-<LOG TYPE>-last-hash
```

5. Restart the CASP server.

13. Web Interface

CASP is provided with a web interface that can be used for the creation of accounts, users, and vaults, and managing quorum operations.

The CASP Web UI is installed automatically with the [CASP RPM Installation](#). If you need to install it manually, see [Appendix B: Web Interface Manual Installation](#).

13.1. CASP Initialization

The first time that you open the CASP interface it guides you through creation of your initial account.

13.1.1. Access the UI

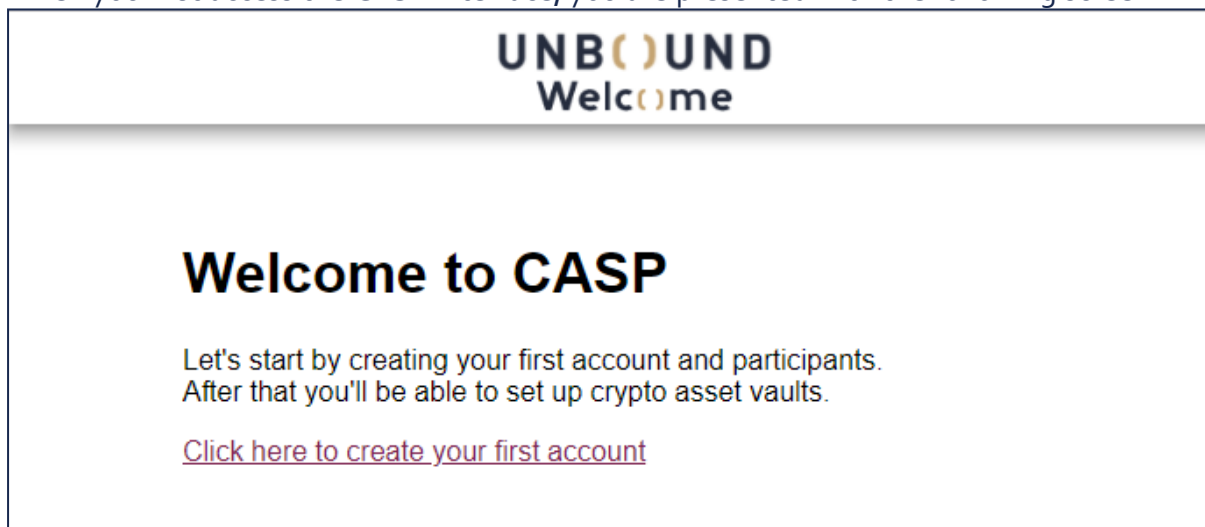
Access the CASP web interface by navigating to: `https://<casp_ip>/caspu1`. The default username is `so` and the default password is `casp`.

An option is provided on the login screen to remember your login for the session. This feature allows the user to refresh the page without having to subsequently log in.

For information about setting up SSO access, see [Single Sign-On](#).

13.1.2. Create an Account

When you first access the CASP interface, you are presented with the following screen.



Click the link and then enter the name of your account.

New account

Name

☐ Global account

Cancel Create account

Note

An option is provided to make this a **Global account**. Participants from global accounts can be included in admin and approval groups for vaults from regular (non-global) accounts.

13.1.3. Create a Vault

Click **Vaults** to access the vaults screen.

unbound
security
[CORE CASP]

Accounts

Users

Participants

Vaults

Data collectors

Templates

Operations

Reports

System

so@CASP

Vaults

There are no vaults in this account. Click *Add vault* to create your first vault.

A vault is a secure container for the cryptographic material used to protect a crypto asset, such as the seed or private key.

To learn more about vaults [click here](#).

Add vault ?

You must create participants before you can create a vault. Click **Participants** from the menu to access that screen.

13.1.4. Add Participants

Add participants as described in [Participants](#). Once all participants are activated, you can create [Vaults](#).

13.2. Accounts

This screen provides information about your accounts, including the pending and total number of participants, vaults and operations.

Name	Creation time	Global	Participants (inactive / total)	Vaults (pending / total)	Operations (pending / total)
CASP	7/29/2021, 9:48:42 AM	no	0 / 1	0 / 0	0 / 0

13.3. Users

This screen lists all users for the account displayed at the top-right of the screen.

Name	Creation time	Modification time	Role	Status	Email
so	7/29/2021, 9:48:34 AM	7/29/2021, 9:48:42 AM	Super user	Active	

CASP has two types of users, called *operators* and *participants*. Use the **Users** screen to manage operators and use the **Participants** screen to manage participants. See [CASP Operators and Participants](#) for more information.

Clicking on **Create** opens a screen to create an *operator*. See [CASP Roles](#) for a description of the **Roles** column.

Creation of an **Operator** has two important options that can be enabled:

1. *Can participate in operation approval for account x* - this operator can be included in admin and approval groups for the vault policy.
2. *Use two-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.

A menu is provided at the end of each user row. Clicking the menu icon provides functions, such as generating a new activation code for a pending user, or replacing a phone for a participant. A complete list of functions is shown in the following table.

User Type	Pending Status Functions	Active Status Functions
Operator	<ul style="list-style-type: none">• Edit• Reset password• Create new activation code• Delete• Show ID• Export participant data	<ul style="list-style-type: none">• Edit• Reset password• Suspend• Revoke• Delete• Show ID• Export participant data

13.4. Participants

On the **Participants** screen, click **Create**. The following screen opens. See [CASP Operators and Participants](#) for more information about the different types of users in CASP.

The screenshot shows the Unbound Security CORE CASP interface. On the left is a dark sidebar with navigation links: Accounts, Users, Participants (highlighted), Vaults, Data collectors, Templates, Operations, Reports, and System. The main area is titled 'Participants' and contains a search bar, a '+ Create' button, and a table with columns 'Active operation' and 'Email'. A modal form titled 'New participant' is open in the center. The form has fields for 'Name' (John Doe) and 'Email' (john@johndoe.com), an 'Offline' checkbox, and 'Cancel' and 'Create participant' buttons. The bottom of the interface shows 'Items per page: 10' and '1 - 1 of 1'.

Enter the name and email and click **Create participant**. There is a checkbox to define the participant as an [Offline Participants](#).

Note

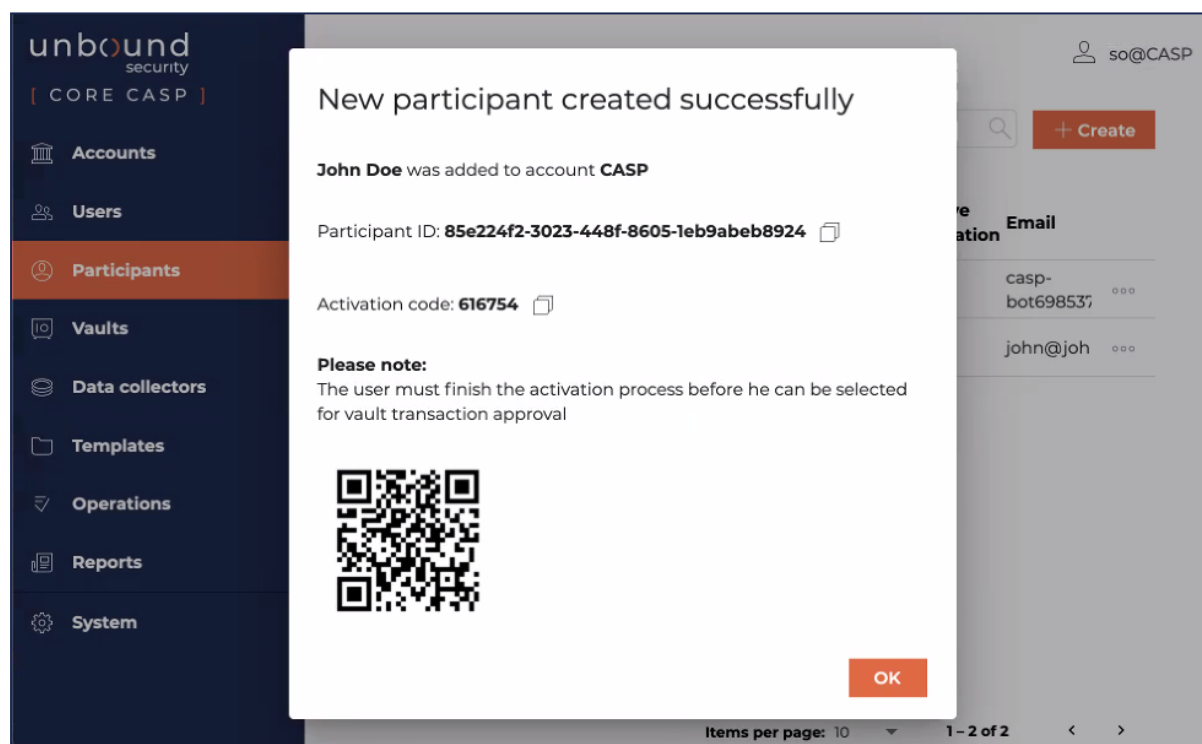
Participants are added to the account shown in the top-right part of the screen, with the format <user>@<account>.

A menu is provided at the end of each user row. Clicking the menu icon provides functions, such as generating a new activation code for a pending participant, or replacing a phone for a participant. A complete list of functions is shown in the following table.

User Type	Pending Status Functions	Active Status Functions
Participant	<ul style="list-style-type: none">EditCreate new activation codeDeleteShow IDExport participant data	<ul style="list-style-type: none">EditReplace phoneSuspendRevokeShow IDExport participant data

Participant Activation

When you create a participant, a screen opens with the activation information for that participant. The participant can scan the QR code with the mobile app (The app has CASP embedded in it that recognizes the QR code). Alternatively, you can send the participant the ID and activation code, or even a screenshot of the QR code.



Note

The participant status remains *Pending* until activated. While pending activation, participants may not be added to a vault.

Repeat this process to add all the relevant participants. Once they are all activated, you can create [Vaults](#).

13.5. Vaults

This screen lists all vaults associated with the account displayed at the top-right of the screen.

Name	Creation time	Modification time	Offline	Wallet type	Description	Status	Pending operations
testVault	7/29/2021, 10:17:52 AM	7/29/2021, 10:18:00 AM	No	Ethereum Ropsten Testnet		Ready	0

Click a row to see the details of the corresponding vault, including any assets contained in the vault and a list of the latest operations executed on the vault.

For example, the vault in the following image was just created. The **Assets** pane shows that it has multiple assets, but no funds. The **Recent operations** pane shows the join operations used when the vault was created.

Created	Kind	Status	Description
Jul 29, 2021 10:23	JOIN_POLICY_VAULT	Completed	Request participants to join policy group for new vault 'riskVault'
Jul 29, 2021 10:23	JOIN_VAULT_ADMIN_QUORUM	Completed	Request participants to join admin group for new vault 'riskVault'

From the main **Vaults** screen, click **Create** to create a new vault. Options are provided to create a [Simple Vault](#) or a [Risk-Based Policy Vault](#).

Note

After creating a vault, the following cannot be changed:

- Add/remove policies
- Add/remove approval groups
- Enable/Disable offline status of a vault, policy, or group

Note

You can create vaults with the same name as long as one is a simple vault and one is a risk-based policy vault. You can change the name of the vault after it is created.

13.5.1. Simple Vault

Simple vaults provide a simple quorum structure for approvals.

New vault

Name

testVault|

Wallet type

Ethereum Ropsten Testnet

Description

☒ Support sub-accounts with hierarchical address generation (BIP-44)

First sub-account name

Account 0

☐ This is an offline vault

Approval groups

Name	Members	Required		
A		0		
B		0		

* Groups are prompted for approvals in the order that they appear here.

+ Add group

Cancel

Create vault

This screen has the following options:

- Name - enter the vault name.
- Wallet type- select the type of wallet.
- Description - (optional) description of the vault.
- Support sub-accounts (BIP44) - select if you want to have sub-accounts.
- Offline - select if you want this vault to be offline. [Offline Vaults](#) must have one approval group that has only [Offline Participants](#).
- Approval groups - define your MofN structure from the available pool of participants.

Note

When there are multiple groups, the order of the groups determines when they are notified about approval requests. For example, group "A" gets notified, and then after enough approvals are received from group "A", group "B" is notified. The order does **not** prevent anyone from group "B" manually checking if any approval requests exist.

Tip

It is recommended that groups have more participants than the required minimum number for approval. This setup allows replacing a participant, which may be necessary if someone loses their phone or someone leaves the company.

13.5.2. Risk-Based Policy Vault

Risk-Based Policy vaults provide complex policy management, including flexible policy groups depending on transaction amounts and time of day. Creating a risk-based policy vault opens a wizard that guides you through the creation process.

Create a vault

1 General info

2 Admin groups

3 Attributes

4 Policies

5 Done

Vault name

riskVault

Wallet type

Ethereum Ropsten Testnet

Description

☒ Support sub-accounts with hierarchical address generation (BIP-44)

First sub-account name

Account 0

☐ This is an offline vault

Cancel

Next

Step 1: General info

This screen has the following options:

- Name - enter the vault name.
- Wallet type- select the type of wallet.
- Description - (optional) description of the vault.
- Support sub-account (BIP44) - select if you want to have sub-accounts.
- Offline - select if you want this vault to be offline. [Offline Vaults](#) must have one approval group that has only [Offline Participants](#).

Step 2: Admin groups

This screen asks you to create administration groups for your vault. Any policy change requires approval from these admin groups.

Note

When there are multiple groups, the order of the groups determines when they are notified about approval requests. For example, group "A" gets notified, and then after enough approvals are received from group "A", group "B" is notified. The order does **not** prevent anyone from group "B" manually checking if any approval requests exist.

Tip

It is recommended that any quorum group has more participants than the required minimum number for approval. This setup allows replacing a participant, which may be necessary if someone loses their phone or someone leaves the company.

Step 3: Attributes

Add attributes (see [Templates](#)) and [Data Collectors](#) to the vault.

Attributes provide a way to add static information to a vault that is sent with all signing requests. Attributes are first defined as a template (see [Templates](#)) and then assigned a value during this stage of vault creation.

Data collectors are defined on the [Data Collectors](#) screen and provide dynamic data that is sent with all signing requests.

Step 4: Policies

You can add approval and/or rejection policies. Each policy has a set of associated conditions that determine when the policy applies and which quorum groups are used for it. Approval policies determine under which conditions a signing request is sent for approval to the quorum. Rejection policies define a set of conditions under which the signing request is rejected.

Each policy has these tabs:

1. General - specify the name of the policy. It is recommended to use a descriptive name, such as "After hours approvals" or "High value transactions".
2. Approval groups (only for approval policies) - add your relevant quorum groups.

Note

When there are multiple groups, the order of the groups determines when they are notified about approval requests. For example, group "A" gets notified, and then after enough approvals are received from group "A", group "B" is notified. The order does **not** prevent anyone from group "B" manually checking if any approval requests exist.

3. Whitelist - create a static whitelist or a derived (dynamic) whitelist. See [Whitelisting](#) for more information.
4. Asset limits - set minimum or maximum amount limits per asset type.
5. Time limits - specify a time period and days of the week when the policy is valid.

For each withdrawal request, the first policy with matching rules from the enabled policies is used. You can drag and drop policies to reorder them.

13.5.3. Vault Backup

A vault can be backed up with this procedure:

1. On the Vaults screen, click on the menu icon next to the vault name.
2. Select **More > Backup**.
3. If the vault has sub-accounts, it prompts you to select the account.

Vault key backup

Vault
test

This is a BIP-44, multi-account vault.
Backup is only supported individually per coin/account

Account to backup
Account 0

Cancel Download backup

4. Click **Download backup**.
5. The backup is saved to the browser's downloads folder.

13.5.4. BIP32 and BIP44

The CASP implementation of hierarchical deterministic ("HD") wallets is based on the de-facto standards [BIP32](#) and [BIP44](#). This section provides some of the features of each of these standards to help you decide which one to use.

Hierarchical deterministic wallets support multiple keypair chains that are derived from a single root. Each wallet has a hierarchy of keypair chains.

- For BIP32 - the hierarchy is completely customizable.
- For BIP44 - the hierarchy uses a standard format and names. The hierarchy can be generated knowing only the seed.

BIP44 has the format: `m / purpose' / coin_type' / account' / change / address_index`

- **m** - master seed.
- **purpose** - Purpose is a constant set to 44' (or 0x8000002C) following the BIP43 recommendation. It indicates that the subtree of this node is used according to this specification.
- **coin type** - a constant, set for each cryptocurrency.
- **account** - This level splits the key space into independent user identities, so the wallet never mixes the coins across different accounts.
- **change** - Constant 0 is used for external chain and constant 1 for internal chain (also known as change addresses). External chain is used for addresses that are meant to be visible outside of the wallet (e.g. for receiving payments). Internal chain is used for addresses which are not meant to be visible outside of the wallet and is used for return transaction change.
- **address_index** - Addresses are numbered from index 0 in sequentially increasing manner.

In general, BIP32 is used for Ethereum and BIP44 is used for Bitcoin.

13.6. Data Collectors

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

Data collectors

Data collectors are independent web services that calculate policy-related attributes for transaction signing.

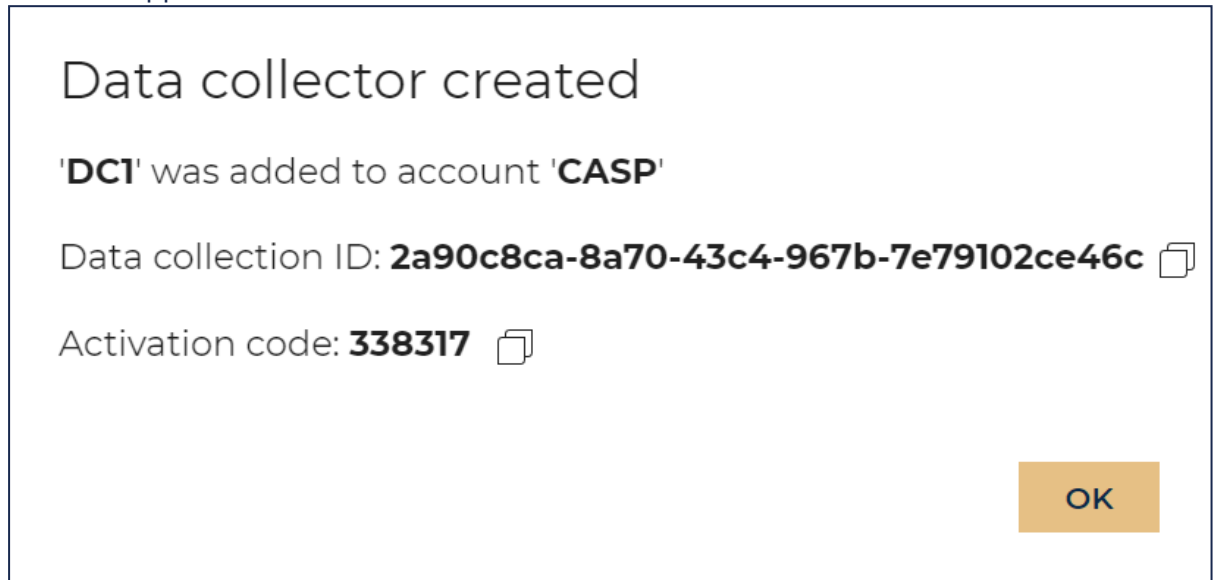
Name	Creation time	Modification time	Description	Status	Attribute template group
collector1	7/29/2021, 9:49:02 AM	7/29/2021, 9:49:02 AM		Active	group1

Items per page: 10 1 - 1 of 1

Unlike participants, which can be human and require no development, data collectors by definition require development by the customer. See the [CASP Java SDK](#) in the CORE Developers Guide for more information on creating the data collector client.

To create a data collector:

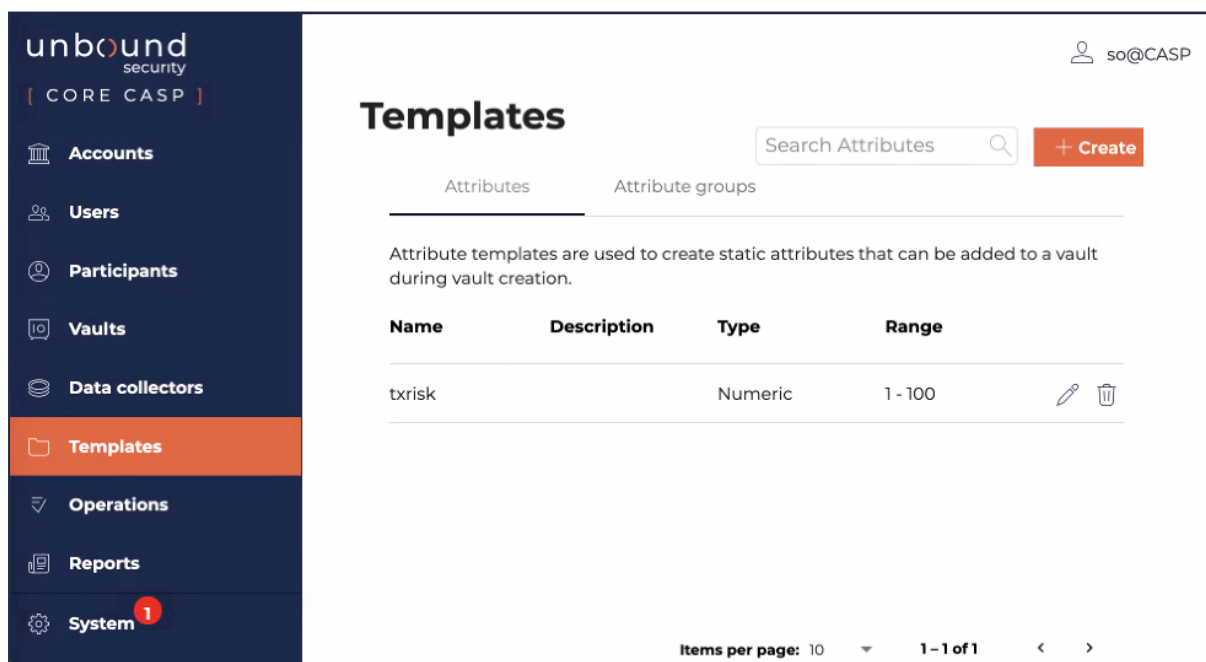
1. Select the **Data Collectors** screen from the left menu.
 2. Click **Add data collector**.
 3. Enter a name and description, and then select the relevant attribute template group.
 4. Click **Create**.
- A screen appears with the details of the data collector.



5. Use the *data collection ID* and *activation code* to set up the client that handles data collection. See the [CORE Developers Guide](#) for more information on creating the data collector client.

13.7. Templates

This screen enables creation of both **attribute templates** and **attribute template groups**.



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.

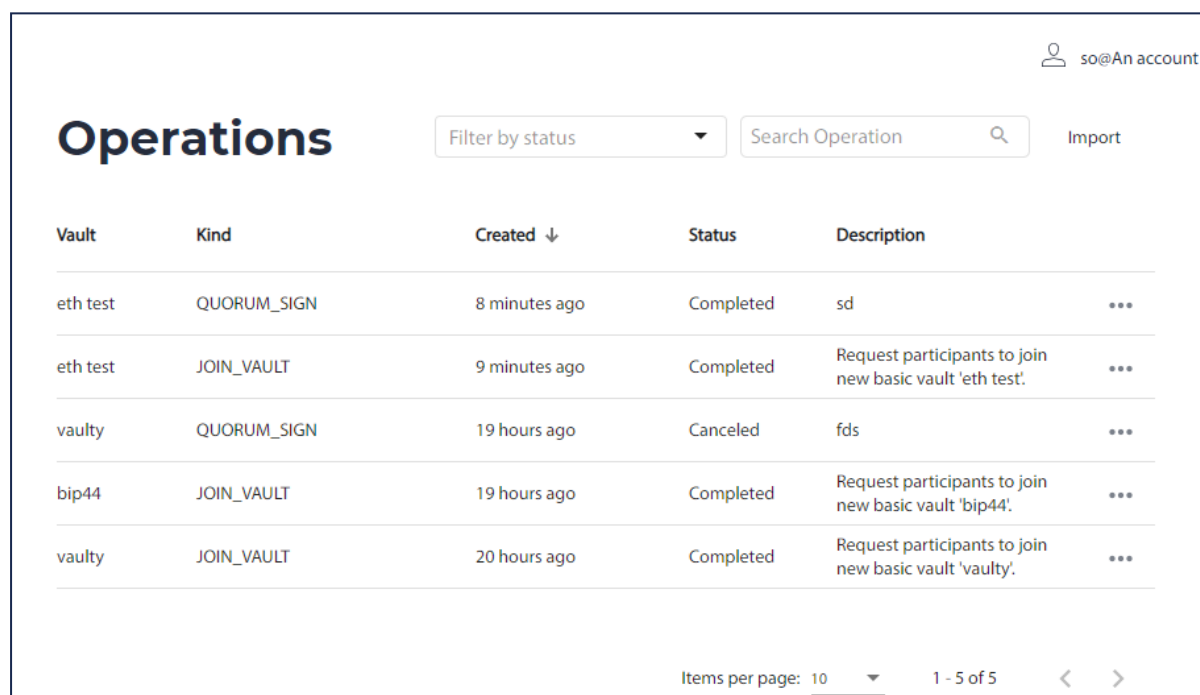
Use this screen to create attribute templates along with their associated limits. For example, create a numerical attribute with a minimum of 2 and a maximum of 1000.

Attribute templates can be grouped together into attribute template groups. These groups are needed for the data collectors.

After creating attribute templates, you can create a [Risk-Based Policy Vault](#) and assign a value to the attributes. During the vault creation flow, there is a screen that allows you to add the attributes and assign values to them.


13.8. Operations

This screen lists all approval quorum operations for the current account.



Vault	Kind	Created ↓	Status	Description	
eth test	QUORUM_SIGN	8 minutes ago	Completed	sd	...
eth test	JOIN_VAULT	9 minutes ago	Completed	Request participants to join new basic vault 'eth test'.	...
vaulty	QUORUM_SIGN	19 hours ago	Canceled	fds	...
bip44	JOIN_VAULT	19 hours ago	Completed	Request participants to join new basic vault 'bip44'.	...
vaulty	JOIN_VAULT	20 hours ago	Completed	Request participants to join new basic vault 'vaulty'.	...

Clicking on the menu next to an operation and selecting **Show info** opens a screen with details about the operation, such as the following for a sign operation:

 so@An account

[Operations](#) / a290e0f9-a6f7-49db-bb36-66dfc7369486

Operation a290e0

sd

QUORUM_SIGN | Completed

Summary

Status:	Completed
Vault:	eth test
Created at:	Tuesday, May 5, 2020 at 1:16 PM
Value:	0.001 ETH
Recipient:	0xFB417294880e0628143E5b1016e9d5bb6A50a7FA

Links

- [Transaction 0xbee2...5da1](#)

Approvals

Group	Approved by	Not approved by	Status
A (1 / 1)	cold part		Approved

Data

▼ request:

- amount: 0.001
- asset: "ETH"
- recipientAddress: "0xFB417294880e0628143E5b1016e9d5bb6A50a7FA"
- fee: "MEDIUM"
- description: "sd"

▼ txs:

▼ 0:

serialized:

"0xf86a80843b9aca0082520894fb417294880e0628143e5b1016e9d5bb6a50a7fa87038d7ea4c68000802aa0b59d4361d186c33385f33495f73d89c361d364644613c8171693364d1834485e330c474373648d8543"

13.8.1. Transaction fee calculation

13.8.1.1. Bitcoin

For bitcoin, fees are calculated using network fee estimates per KB, based on averages from the blockchain.

For example:

```
"high_fee_per_kb": 25555,
"medium_fee_per_kb": 25000,
"low_fee_per_kb": 15000,
```

When the user selects the fee preference, HIGH, MEDIUM or LOW, the matching `x_fee_per_kb` is multiplied by the transaction size in KB to get the actual fee.

Note

The fee is not configurable for Bitcoin.

13.8.1.2. Ethereum

Ethereum transaction fees are calculated by gas units multiplied by price per unit. When sending a transaction, senders can decide the limit of gas units that they are willing to pay and what price per unit they are willing to pay. The actual fee is $\{\text{number of gas units used}\} * \{\text{price per unit}\}$. The number of units that was used might be lower than the limit set by the sender.

To use HIGH, MEDIUM, and LOW fee preferences, the user must configure a set of multipliers in the *production.yaml* file.

For example:

```
gasPriceFeeMultipliers: {  
  MEDIUM: 1,  
  LOW: 0.8,  
  HIGH: 1.5  
}
```

When sending a transaction, the relevant multiplier is selected and applied on the average network gas price. The multiplier constant must be configured, otherwise the fee calculation fails and the transaction fails on build.

13.9. Reports

CASP empowers custodians to adhere to governance, risk management and compliance (GRC) guidelines by providing operations users a simple way to export data in CSV format from the user interface. The **Reports** screen provides viewing, filtering, and downloading reports for Vaults, Users, Accounts, Operations and Audit.

For example:

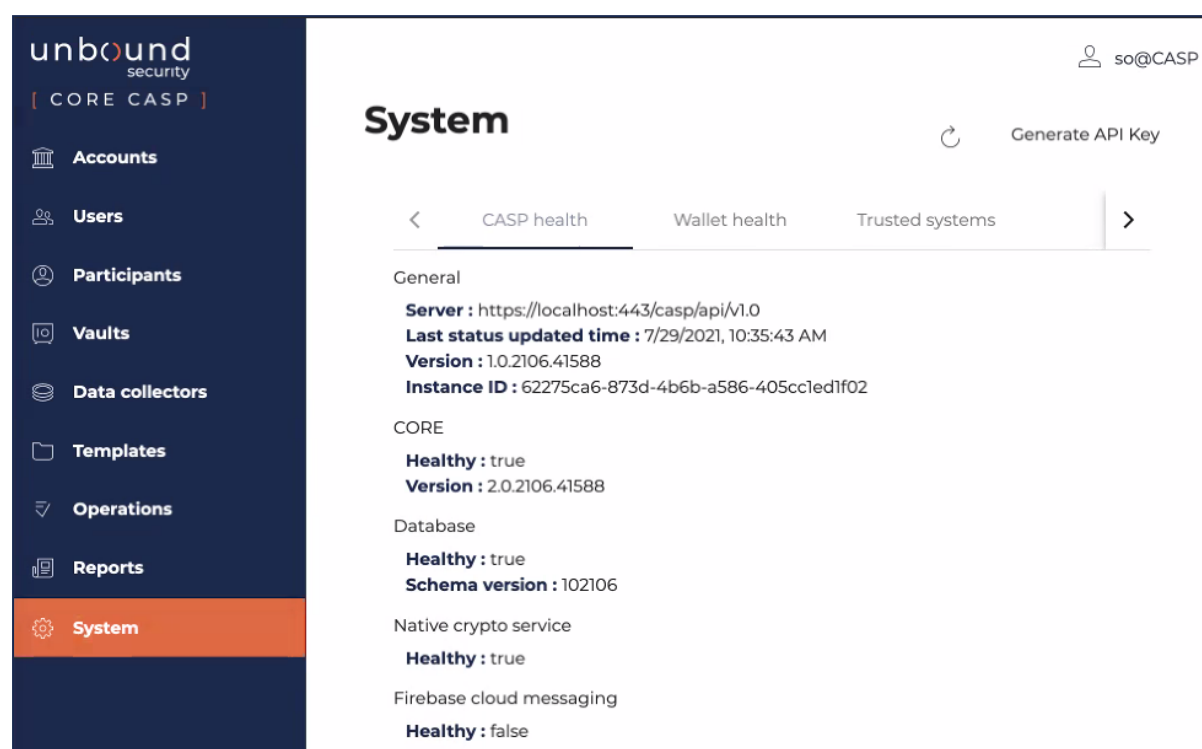
- From the **Operations** tab - a user (such as an accountant) can view, filter, and then download transaction data, which can be shared with the end customers, regulators, with a 3rd party trustee, etc.
- From the **Users** tab - a user can view, filter and then download the list of users and their activation status.
- From the **Vaults** tab - a user (such as a CISO or a supervisor) can view, filter and then download data of the various vaults that are in CASP. This data is most important for custodians that manage thousands of vaults of different customers. A user can share the exported data with any party, such as the end customers, regulators, etc.

Notes

1. Times in output reports are in UTC.
2. If a time value is unknown, it is displayed as *1/1/1970 0:00*.
3. If your report contains non-English characters and you are trying to open the CSV file in Microsoft Excel, you may need to use the import data wizard to import the data with the correct character encoding.

13.10. System

This screen provides information about CASP health, Wallet health, and Trusted systems.



The screen has sections for:

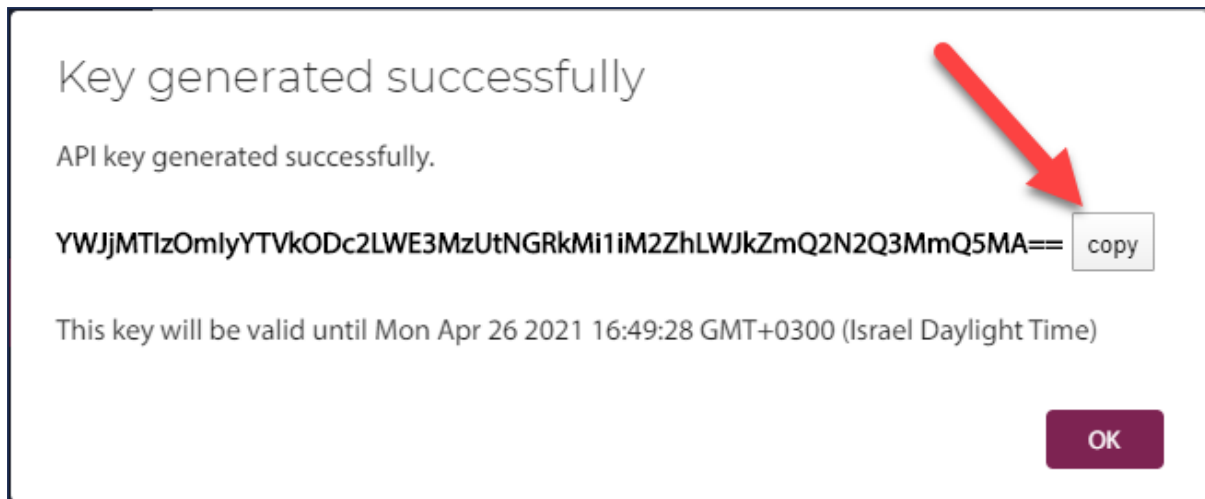
1. Alerts - if there are any issues that may prevent the CASP system from operating properly, they appear at the top. For example, in the previous image the Firebase and Ethereum tokens were not specified.
2. CASP health - shows status information about the components in the CASP system.
3. Wallet health - shows the status of the [Built-in Wallets](#).
4. Trusted systems - shows information about the trusted systems and enables adding and removing these systems. See [Trusted Systems](#) for more information.

There is a button, called *Generate API Key*, that generates a new API key.

Note

When creating an API key, you must define a user and the user's role. Information about roles can be found in [CASP Roles](#).

After entering the key details, the resulting screen provides a *copy* button to copy the API key to the clipboard.



13.11. Participant Management

The following sections detail procedures for common operations needed when managing participants.

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 leaves the company or a specific vault \(suspend\)](#)
6. [Participant leaves the company or a specific vault \(revoke\)](#)
7. [Participant replaces a phone \(reactivate\)](#)
8. [BOT becomes unavailable](#)
9. [Replace a Participant](#)

13.11.1. Create a participant

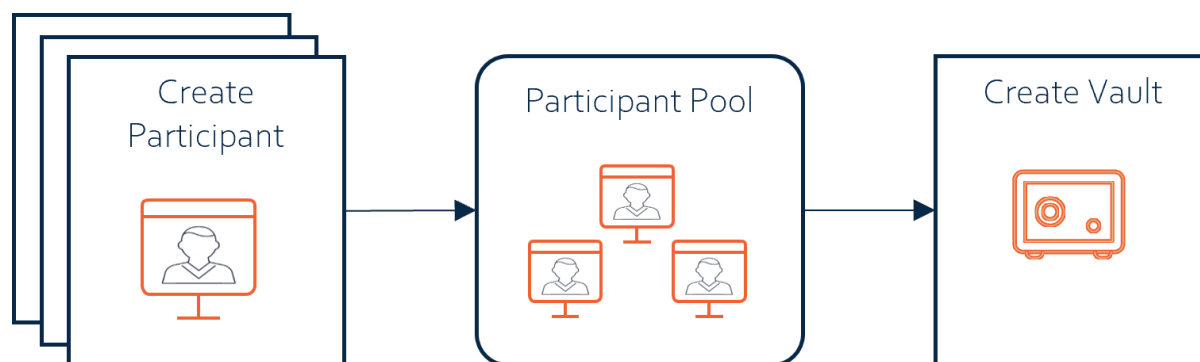
A participant is created in a specific account. On the **Participants** screen, click **Create**.

13.11.2. Update participant details

You can update a participant's details, including the name and email address, by clicking the menu icon in the [Participants](#) screen.

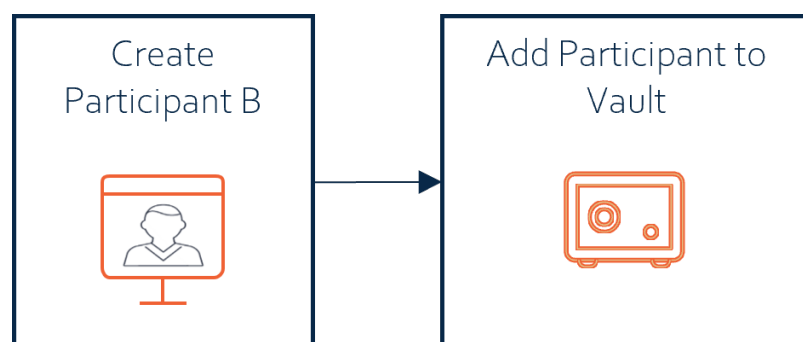
You can also use this endpoint to globally set the participant status to SUSPENDED or REVOKED. A suspended or revoked participant cannot perform any action on any operation, such as approving a sign request.

13.11.3. 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 on the [Vaults](#) screen. 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.

13.11.4. 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 on the [Vaults](#) screen.
- Step 3: The required quorum group members must approve this operation.
- Step 4: The participant must approve being added to the vault.

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

A participant may need to be suspended in different situations. For example:

- Regulatory hold - for some regulatory reason, the participant needs to be prevented from performing any actions.

To suspend a participant:

1. On the **Participants** screen, click the options button next to the participant.
2. Select **Suspend**.

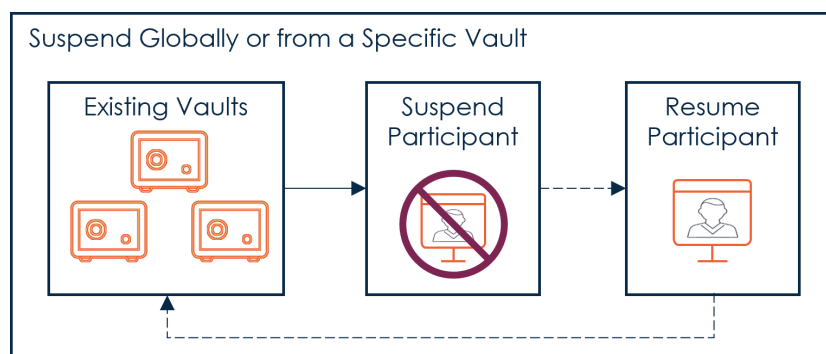
The screenshot shows the 'Participants' screen with a search bar and a '+ Create' button. The table below lists participants:

Name	Creation time	Modification time	Device Type	Offline	Status	Active operations	Email
casp-bot2b57595	11/30/2020, 9:58:15 AM	11/30/2020, 9:58:20 AM	Bot	No	Active	0	casp-bot2b57595k
Joe Blough	12/1/2020, 11:00:01 AM	12/1/2020, 11:00:01 AM		No	Pending	0	

A context menu for 'Joe Blough' is open, showing options: Edit, Replace phone, **Suspend** (highlighted with a hand icon), Revoke, and Show ID. At the bottom, it says 'Items per page: 10' and '1 - 2 of 2'.

Upon successful suspension, the participant **Status** changes from *Active* to *Suspended*.

This participant can be activated in the [Participants](#) screen by selecting the menu icon and then *Resume*.



Note

You receive a warning if suspending a participant violates the quorum policy. For example, if the quorum requires 2 participants for approval, and there are only 2 participants, and you suspend one of them. The best practice is to first add a new participant (for a total of 3) and then one can be suspended.

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

A participant may need to be revoked in several situations, such as:

- Functional job change - the participant changes to a new position and should no longer be a member of the quorum.
- Employee leaves - the employee leaves the company.

To revoke a participant:

1. On the **Participants** screen, click the options button next to the participant.
2. Select **Revoke**.

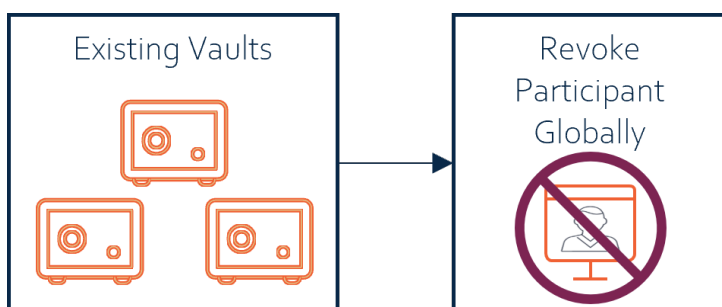
Participants

Search Participant

Name	Creation time	Modification time	Device Type	Offline	Status	Active operations	Email
casp-bot2b57595	11/30/2020, 9:58:15 AM	11/30/2020, 9:58:20 AM	Bot	No	Active	0	casp-bot2b57595k
Joe Blough	12/1/2020, 11:00:01 AM	12/1/2020, 11:00:01 AM		No	Pending	0	

Items per page: 10 1 - 2 of 2

Upon revoking successfully, the participant **Status** changes from *Active* to *Revoked*.



Note

You cannot revoke a participant if it violates the quorum policy. For example, if the quorum requires 2 participants for approval, and there are only 2 participants, you cannot remove either one. First, add a new participant (for a total of 3) and then one can be removed.

13.11.7. Participant replaces a phone (reactivate)

If one of your participants replaces their phone with a new one (such as after losing a phone), use the following procedure:

1. On the **Participants** screen, click the options button next to the participant.
2. Select **Replace phone**.

Participants

Search Participant

Name	Creation time	Modification time	Device Type	Offline	Status	Active operations	Email
casp-bot2b57595	11/30/2020, 9:58:15 AM	11/30/2020, 9:58:20 AM	Bot	No	Active	0	casp-bot2b57595t
Joe Blough	12/1/2020, 11:00:01 AM	12/1/2020, 11:00:01 AM		No	Pending	0	

Items per page: 10 1 - 2 of 2

3. A screen appears that shows a list of all the vaults that the participant is a part of. Select the vaults that you want to reactivate the user in. The default is all vaults.
4. A new activation code appears.

Note

Participants cannot approve/deny operations until they are approved to be re-added to the relevant vaults.

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.

Note

If a participant is added to a vault and the reactivate command is executed **before** the participant approves joining the vault, the command returns an error. The operation to add the participant must first be canceled and then the reactivate can be resubmitted.

13.11.8. BOT becomes unavailable

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

- Step 1: Deactivate the BOT using [Suspend](#) or [Revoke](#).
- Step 2: Run the steps in [Add a participant to an existing vault](#) to activate the new BOT in all of the relevant vaults.

13.11.9. Replace a Participant

If you want to replace one of the participants, use the following procedure.

Warning

Your quorum groups should have more participants than the required minimum number for approval. This setup allows replacing a participant, which may be necessary if someone loses their phone or someone leaves the company.

1. If the participant that you want to add does not yet exist in CASP, then add a new participant.
 - a. On the **Participants** screen, click **Create**.
 - b. Fill in the details and then click **Create participant**.

Wait for the participant to become activated.

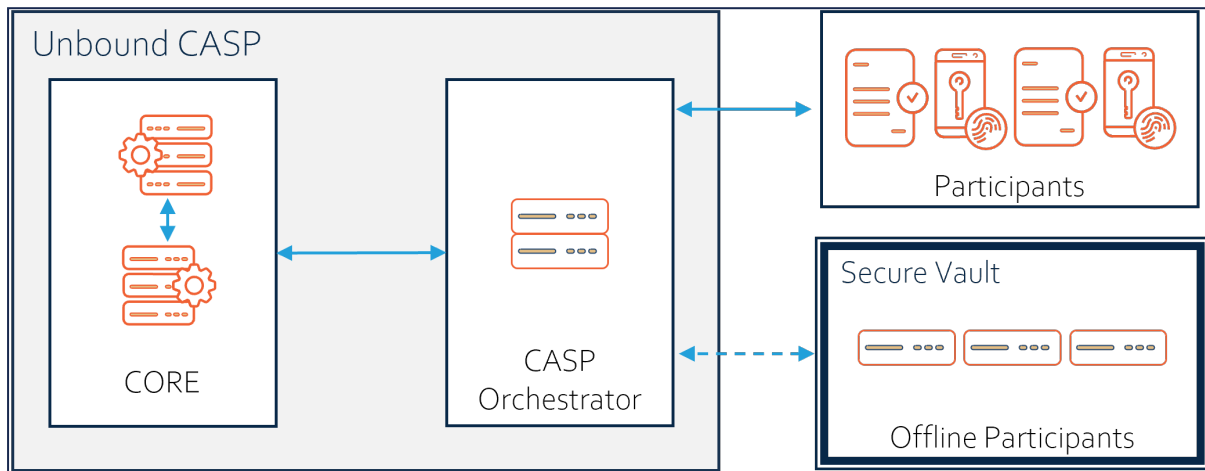
2. Add the participant to the vault.
 - a. On the **Vaults** screen, click the vault name.
A screen opens showing the vault details.
 - b. Click the **Admin groups** tab.
 - c. Select **Add a member** from the bottom-right corner of the pane.
 - d. Fill in the details and then click **Add member**.

The quorum needs to approve adding a new member.

3. Suspend (or revoke) the participant that is being replaced with the instructions in [Participant put on hold globally or in a specific vault \(suspend\)](#).

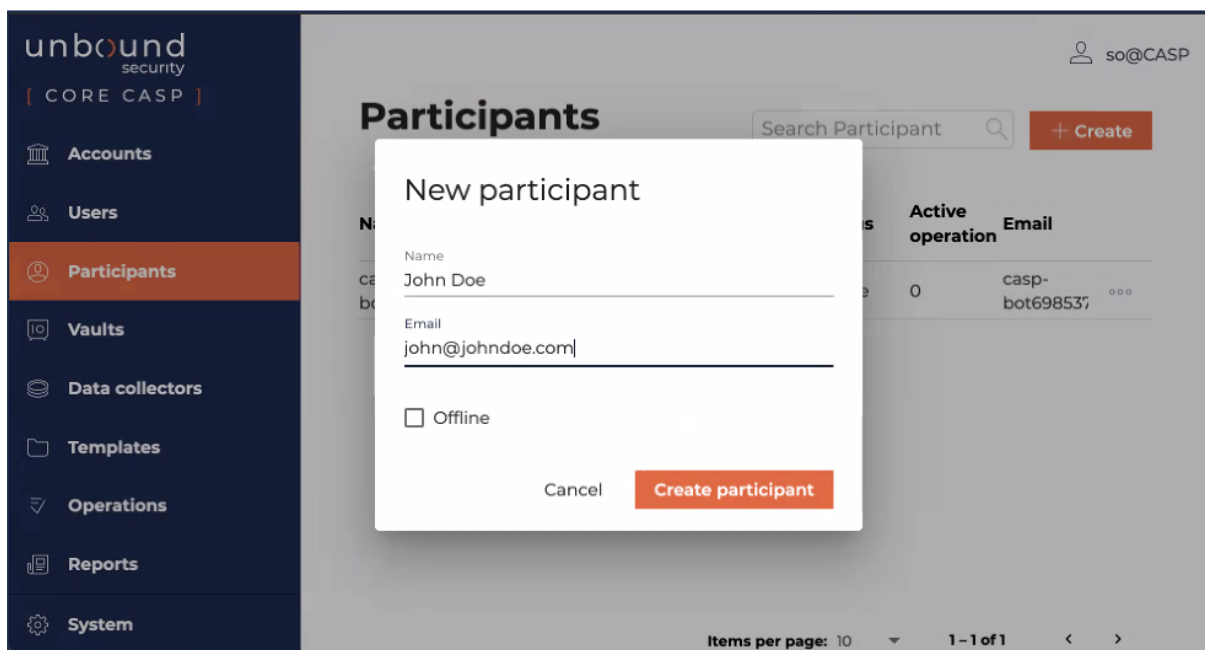
14. Offline Vaults

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 solution where part of the approval process takes places in highly secure areas, such as keeping the offline participants in a guarded vault. An example system is shown in the following figure.



14.1. Offline Participants

When creating a participant, enable offline by clicking the *Offline* checkbox.



Once created, the offline status of a participant cannot be changed.

Note

Offline participants cannot be added to a regular (non-offline) vault after it has been created.

14.2. Offline Vaults

Note

An offline vault must contain at least one approval group that has **only** offline participants.

To create an offline vault:

- For a **Simple** vault, enable *This is an offline vault*.

New vault

Name

testVault|

Wallet type

Ethereum Ropsten Testnet

Description

☒ Support sub-accounts with hierarchical address generation (BIP-44)

First sub-account name

Account 0

☐ This is an offline vault

Approval groups

Name	Members	Required		
A		0		
B		0		

* Groups are prompted for approvals in the order that they appear here.

+ Add group

Cancel

Create vault

- For a **Risk-Based Policy** vault, enable *This is an offline vault*.

Create a vault

1 General info

2 Admin groups

3 Attributes

4 Policies

5 Done

Vault name

riskVault

Wallet type

Ethereum Ropsten Testnet

Description

☒ Support sub-accounts with hierarchical address generation (BIP-44)

First sub-account name

Account 0

☐ This is an offline vault

Cancel

Next

14.3. Offline Policies

Offline policies only apply to risk-based policy vaults. When creating a vault policy, enable offline by clicking *This is an offline policy*. Once created, the offline status of a policy cannot be changed.

15. 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.

After configuring another system to be **trusted**, you can export and import account, participant, vault and operation data.

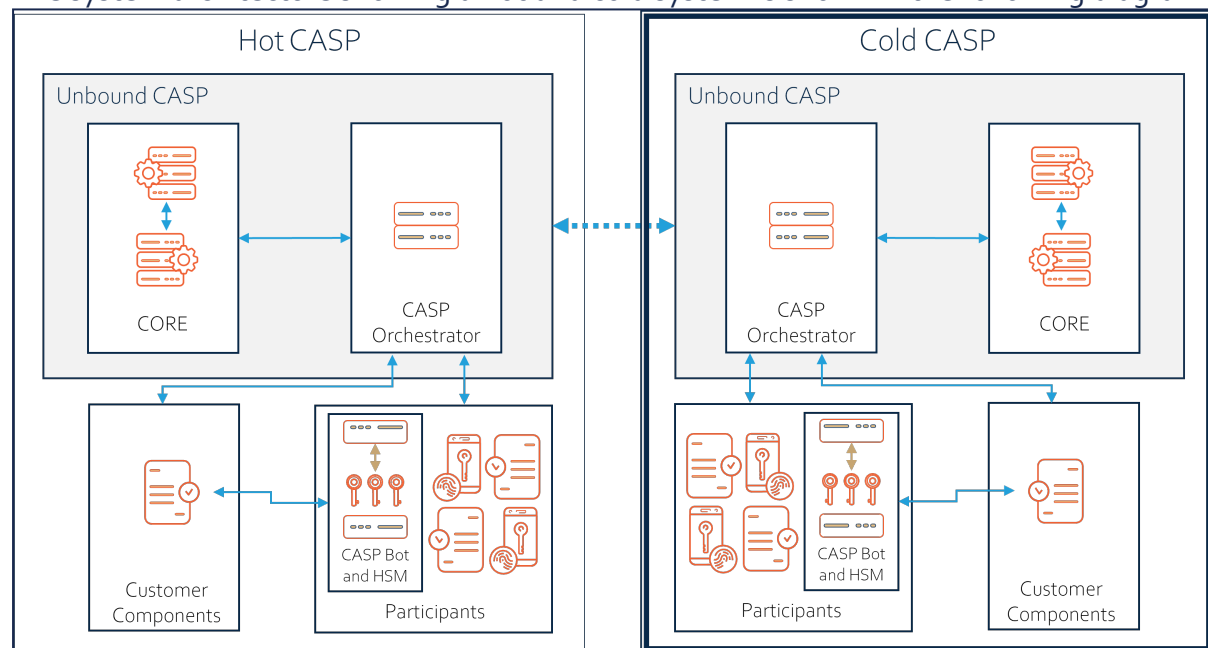
The trusted system architecture uses hot and cold storage vaults. This architecture provides the following benefits:

1. **Secure transactions.** By running key generation ceremonies and storing some of the assets **only** in cold storage, we ensure security of the assets.
2. **Secure backups.** Encrypted backups are created and stored **only** in cold storage.

Once you set up a trusted system you can initiate transactions on the **cold** storage and continue the approval and transaction execution on the **hot** system.

15.1. Sample Trusted System Architecture

The system architecture showing a hot and cold system is shown in the following diagram.



15.2. Configure Trusted Systems

The following sections describe how to configure your trusted systems for use. The procedure involves getting the hot and cold systems to trust each other and then synchronizing accounts, vaults, and participants. These actions consist of exporting data files from one CASP system, manually transferring the data to the other CASP system (such as with a USB flash drive), and importing the data into the 2nd system. Some of the actions require a response from the 2nd system back to the 1st system in a similar manner.

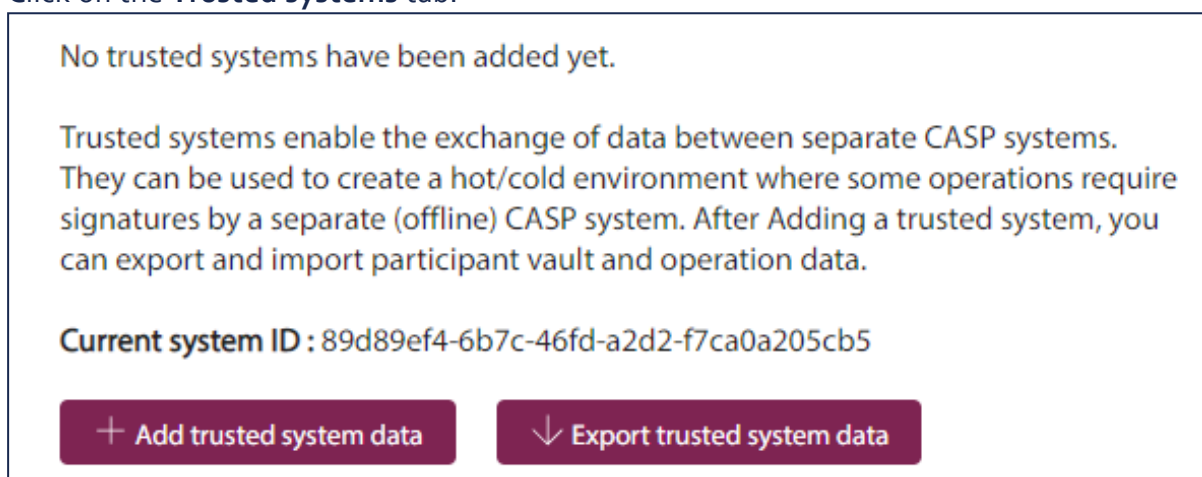
You should have two CASP systems installed and running before starting these procedures. One system is referred to as the **Hot** system and one as the **Cold** system.

15.2.1. Add a Trusted System

CASP can trust other systems. This is a one-way trust, so system A needs to trust system B, and then system B needs to trust system A to establish bidirectional trust.

To add a trusted system:

1. Open the UI for the **Hot** system.
 - a. Click on the **System** menu item.
 - b. Click on the **Trusted systems** tab.



- c. Click **Export trusted system data**.

This action creates a file in your *Downloads* directory with information about the Hot system.
2. Open the UI for the **Cold** system.
 - a. Click on the **System** menu item.
 - b. Click on the **Trusted systems** tab.
 - c. Click **Add trusted system data**.
 - d. Select the file exported from the **Hot** system.
 - e. It is recommended to fill in the *description* field so that you know that this is the **Hot** system.

The **Cold** system now trusts the **Hot** system. Execute the same procedure again, but switch hot and cold to establish trust from cold to hot and hot to cold.

15.2.2. Synchronize Accounts

The following procedure describes how to synchronize an account to the **Hot** system.

To synchronize accounts:

1. In the UI for the **Cold** system, click on the **Accounts** menu item.
2. At the end of the row for the account that you want to synchronize, click the menu icon and select *Export account data*.
3. This action creates a file in your *Downloads* directory with information about the account.
4. In the UI for the **Hot** system, click on the **Accounts** menu item.
5. Click **Import**.
6. Enter the file exported from the **Cold** system and click **Upload**.

The account is now synchronized between the Hot and Cold systems.

15.2.3. Synchronize Participants

If you do not have participants on the **Hot** and **Cold** system, create them. This example uses *UserH* for the participant on the **Hot** system and *UserC* for the participant on the **Cold** system. The following procedure describes how to synchronize the participants between the systems.

To synchronize participants:

1. In the UI for the **Hot** system, click on the **Users** menu item.
2. At the end of the row for *UserH* that you want to synchronize, click the menu icon and select *Export participant data*.
3. This action creates a file in your *Downloads* directory with information about the participant.
4. In the UI for the **Cold** system, click on the **Users** menu item.
5. Click **Import**.
6. Enter the file exported from the **Hot** system and click **Upload**.

The participant, *UserH*, is now synchronized between the **Hot** and **Cold** systems. Perform the same procedure, but exporting *UserC* data from the **Cold** system and importing into the **Hot** system.

15.2.4. Create and Synchronize Vaults

If you do not have a vault on the **Cold** system, create it. Since you synchronized the participants, you can create an approval group for the vault with both *UserH* and *UserC*. When you create a vault on the Cold system, the join request for *UserC* can be approved by the **Cold** participant. The vault needs to be synchronized with the **Hot** system to be able to get the join approval from the Hot participant, *UserH*. After *UserH* joins, the vault needs to be exported from the **Hot** system and imported into the **Cold** system to receive the join information about *UserH*.

The following procedure describes how to synchronize the vault. We assume that *UserC* has already approved the join request.

1. Synchronize **Cold** to **Hot**:
 - a. In the UI for the **Cold** system, click on the **Vaults** menu item.
 - b. At the end of the row for the vault that you want to synchronize, click the menu icon and select *Export vault data*.
 - c. This action creates a file in your *Downloads* directory with information about the vault.
 - d. In the UI for the **Hot** system, click on the **Vaults** menu item.
 - e. Click **Import**.
 - f. Enter the file exported from the **Cold** system and click **Upload**.
 - g. *UserH* can now approve the join request in the **Hot** system.
2. Synchronize **Hot** to **Cold**:
 - a. In the UI for the **Hot** system, click on the **Vaults** menu item.
 - b. At the end of the row for the vault that you want to synchronize, click the menu icon and select *Export vault data*.
 - c. This action creates a file in your *Downloads* directory with information about the vault.
 - d. In the UI for the **Cold** system, click on the **Vaults** menu item.
 - e. Click **Import**.
 - f. Enter the file exported from the **Hot** system and click **Upload**.
 - g. Both *UserC* and *UserH* approved joining the vault, and the *Status* of the vault should display as *Ready*.

The vault is now synchronized between the **Hot** and **Cold** systems.

15.2.5. Synchronize Operations

You may need to synchronize operations to [Sign Transactions](#) or other operations such as adding a new member.

To synchronize operations:

1. In the UI for the **Cold** system, click on the **Operations** menu item.
2. At the end of the row for the operation that you want to synchronize, click the menu icon and select *Export operation data*.
3. This action creates a file in your *Downloads* directory with information about the operation.
4. In the UI for the **Hot** system, click on the **Operations** menu item.
5. Click **Import**.
6. Enter the file exported from the **Cold** system and click **Upload**.

15.3. Sign Transactions

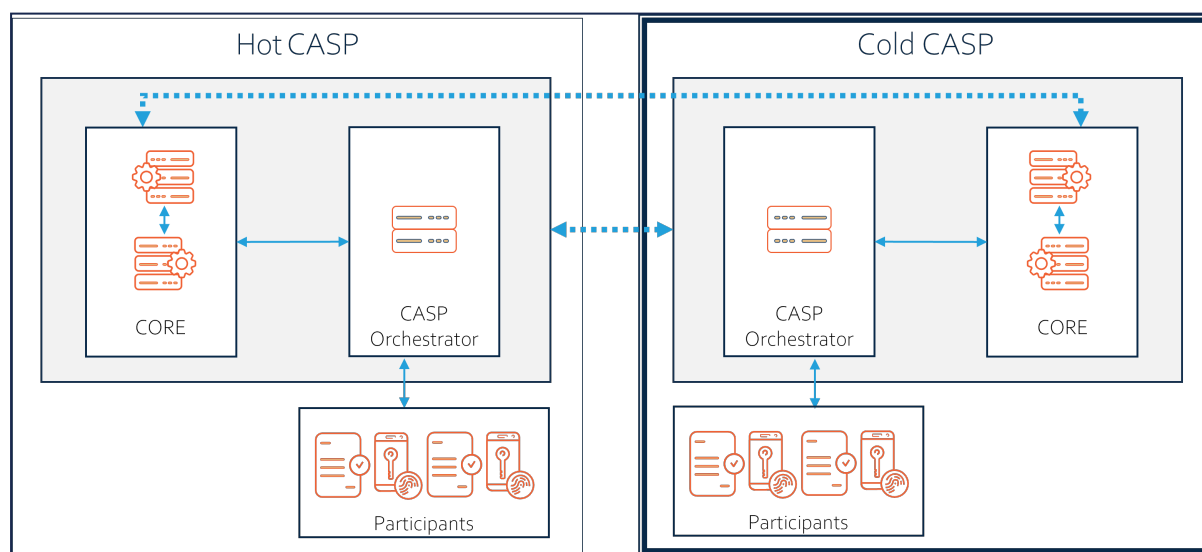
The following procedure describes how to sign a transaction that was created on the **Cold** system on the **Hot** system.

1. Start a signing process on the **Cold** system. Note that for this example *UserC* approves the transaction immediately on the **Cold** system.
 - a. In the UI for the **Cold** system, click on the **Operations** menu item.
 - b. At the end of the row for the operation that you want to synchronize, click the menu icon and select *Export operation data*.
 - c. This action creates a file in your *Downloads* directory with information about the operation.
 - d. In the UI for the **Hot** system, click on the **Operations** menu item.
 - e. Click **Import**.
 - f. Enter the file exported from the **Cold** system and click **Upload**.
 - g. *UserH* can now approve the sign request in the **Hot** system.
2. Synchronize **Hot** to **Cold**:
 - a. In the UI for the **Hot** system, click on the **Operations** menu item.
 - b. At the end of the row for the operation that you want to synchronize, click the menu icon and select *Export operation data*.
 - c. This action creates a file in your *Downloads* directory with information about the operation.
 - d. In the UI for the **Cold** system, click on the **Operations** menu item.
 - e. Click **Import**.
 - f. Enter the file exported from the **Hot** system and click **Upload**.
 - g. Both *UserC* and *UserH* approved the sign request operation, and the *Status* of the vault should display as *Completed*.

The operation is now completed and synchronized between the **Hot** and **Cold** systems.

15.4. (Optional) Synchronize CORE Keys

In some cases you may want to be able to create vaults, start sign operations, and perform the key ceremony in either the **Hot** or **Cold** systems (instead of only in the **Cold** system).



If you need this ability use the following procedure to import the keys from the CORE. In this procedure, C1 refers to the **cold** system (cluster), and C2 refers to the **hot** system (cluster).

Notes

1. This procedure must be run once for the Entry Point server and once for the Partner server.
2. The scripts used in the procedure are in the CORE installation.

To import the keys from the CORE:

1. On the **Hot** system:
 - a. Generate the key encryption key ("KEK").

```
ekm_gen_kek
```

- b. Download the KEK certificate.

```
ekm_get_kek
```

- c. Download the cluster (C2) root CA. Run this command only on the EP. On the Partner use the certificate from the EP.

```
uc1 root_ca -o C2CA.pem
```

- d. Copy the KEK certificate and *C2CA.pem* files to the **Cold** system.

2. On the **Cold** system:
 - a. Generate the key-signing key ("KSK").

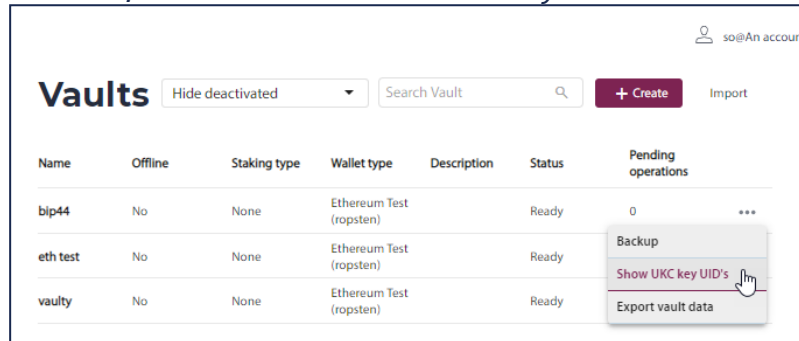
```
ekm_gen_ksk
```

- b. Set *C2CA.pem* as the trusted CA for key wrapping.

```
ekm_set_key_wrap_ca -ca C2CA.pem
```

- c. Wrap the key. The wrapped key output is *K_UID.E1.tar.gz*.

- First, find the key UID. On the **Vaults** screen, click the menu icon next to the vault, and then select *Show CORE key UID's*.



- Next, run the key wrap script for each UID from the previous step.

```
ekm_wrap_key -cert KEK -u UID -p <partition-name>
```

- d. Get the KSK.

```
ekm_get_ksk
```

- e. Download the cluster (C1) root CA. Run this command only on the EP. On the Partner use the certificate from the EP.

```
uc1 root_ca -o C1CA.pem
```

- f. Copy *K_UID.E1.tar.gz*, the KSK certificate, and *C1CA.pem* to **Hot** system.

3. On the **Hot** system:

- a. Set *C1CA.pem* as the trusted CA for key wrapping.

```
ekm_set_key_wrap_ca -ca C1CA.pem
```

- b. Unwrap the KSK.

```
ekm_un_wrap_key -cert E1_KSK -w K_UID.E1.tar.gz -p <partition-name>
```

You can now start sign operations and perform the whole key ceremony on both **Hot** and **Cold** systems.

16. Mobile App

CASP can be integrated into a mobile app to provide participant functionality. An iOS app, called the **Unbound CASP Signing App**, is provided as a reference for this functionality. For the list of system requirements, see [Client Requirements](#).

Note

Every time the app asks for an approval it authenticates with either Face ID or Touch ID.

Procedures are provided describing the necessary steps from when you first open the app, through approving a vault, and approving operations.

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.

See [Participant Flows](#) for more information about how the mobile app is used in actual work flows.

16.1. Prerequisites

The **Unbound CASP Signing App** can be found and installed from the Apple App Store using [this link](#).

The CASP app requires one of the following:

- Device PIN
- Enrollment in biometric authentication (such as Face ID or Touch ID)

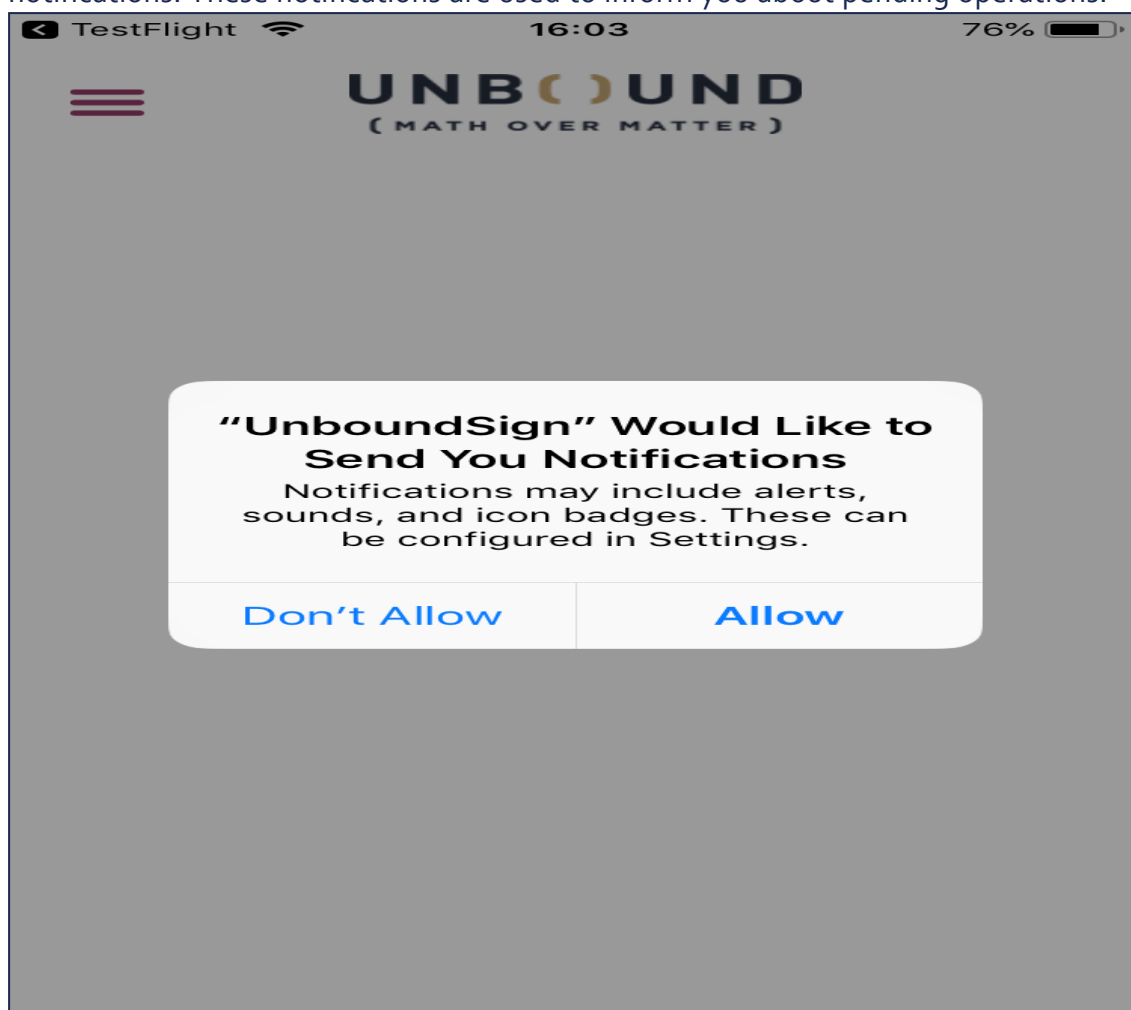
Note

Users can safely change their pin or biometric authentication, but one or the other must always exist on the device.

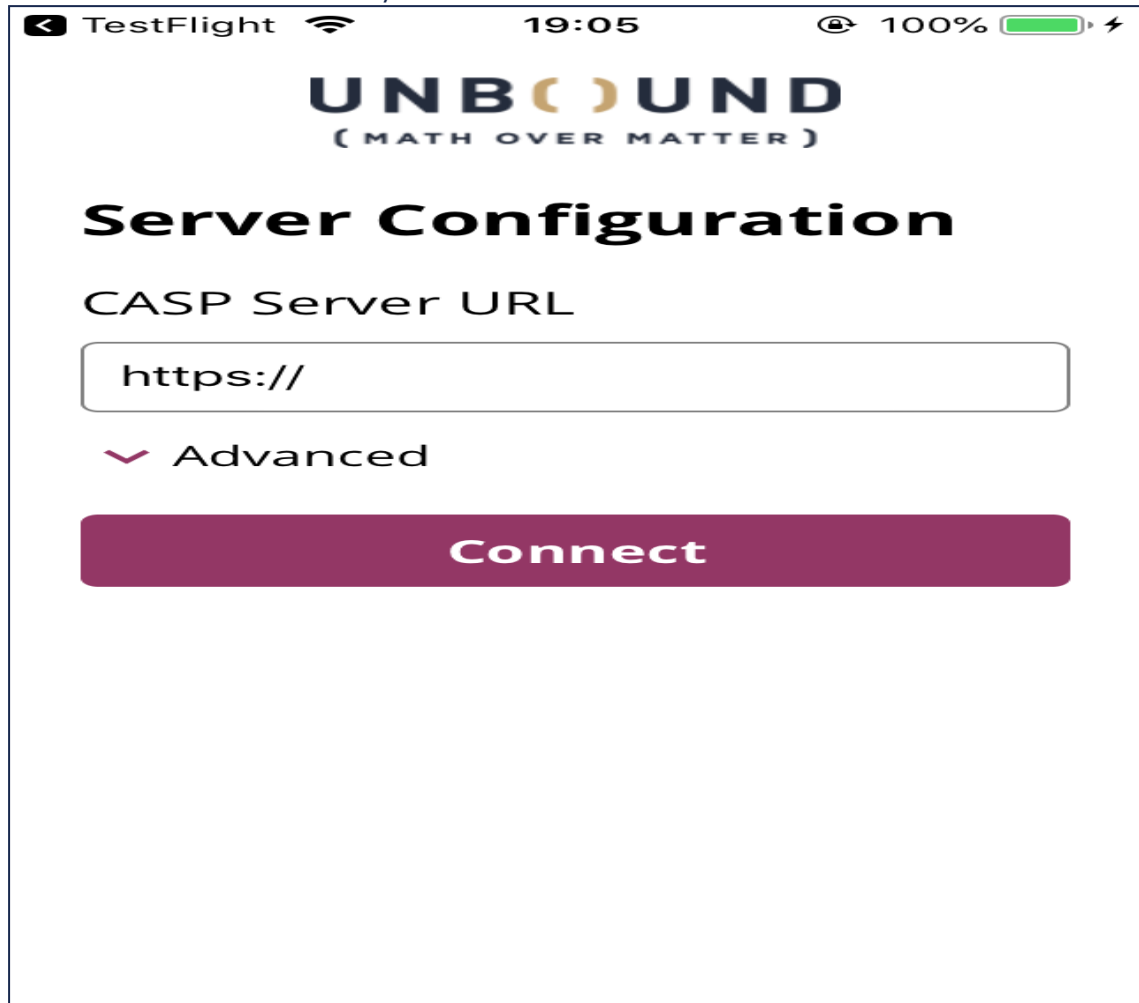
16.2. App Setup

When you first open the app, set up the notifications and the CASP Server:

1. **Notifications** - When you first start the app it prompts you to accept receiving notifications. These notifications are used to inform you about pending operations.



2. **CASP Server** - Enter the URL of the CASP server. The URL has the format:
`http(s)://<server url>`
You can obtain the url from your CASP administrator.



TestFlight 19:05 100%

UNBOUND
(MATH OVER MATTER)

Server Configuration

CASP Server URL

https://

✓ Advanced

Connect

16.3. Participant Activation


The user is activated by entering a Participant ID and an Activation code. The Participant ID can be scanned in by the app.

1. Tap Scan QR Code.

Partner 14:07 100%

UNBOUND
(MATH OVER MATTER)

Participant Activation

 **Scan QR Code**

OR

Enter Participant ID

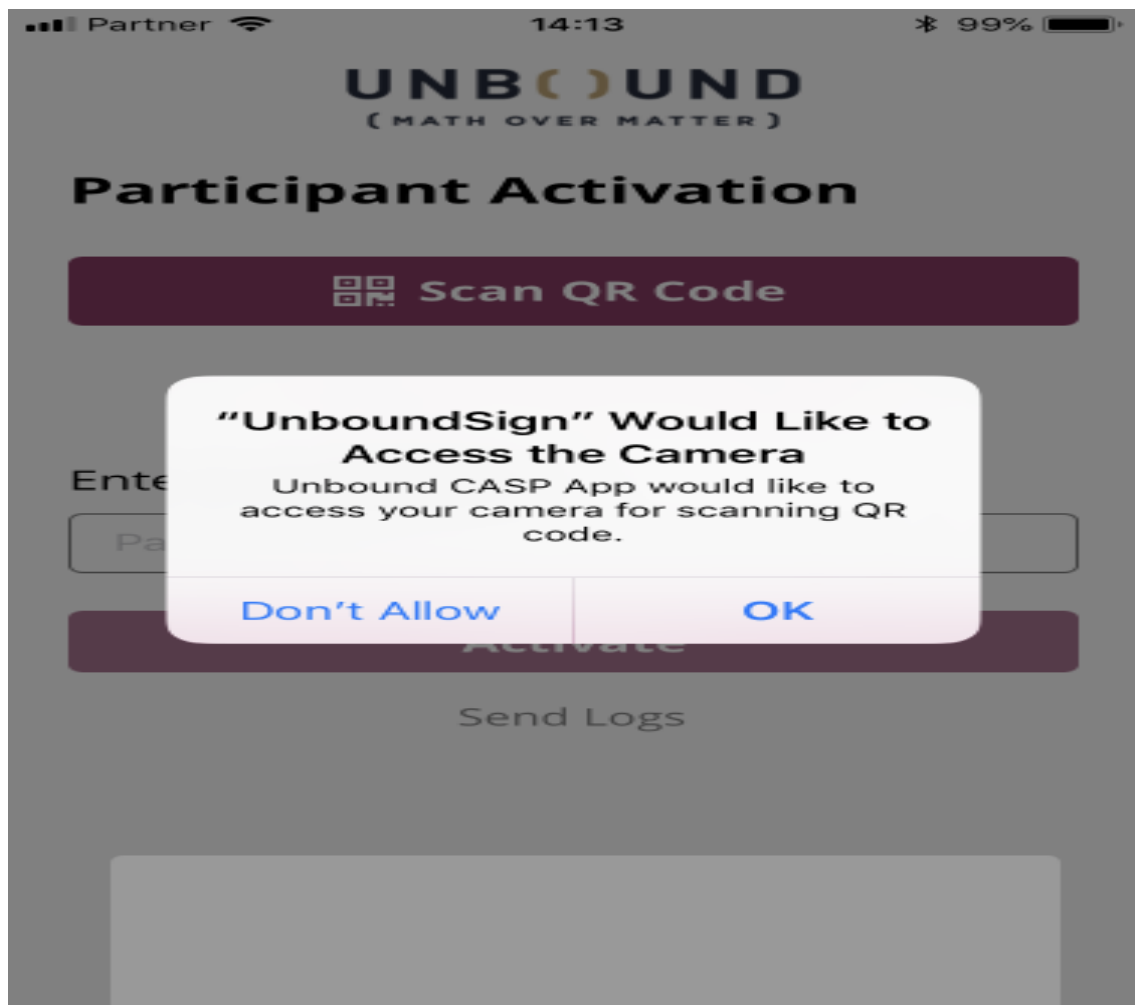
Participant ID

Activate

[Send Logs](#)

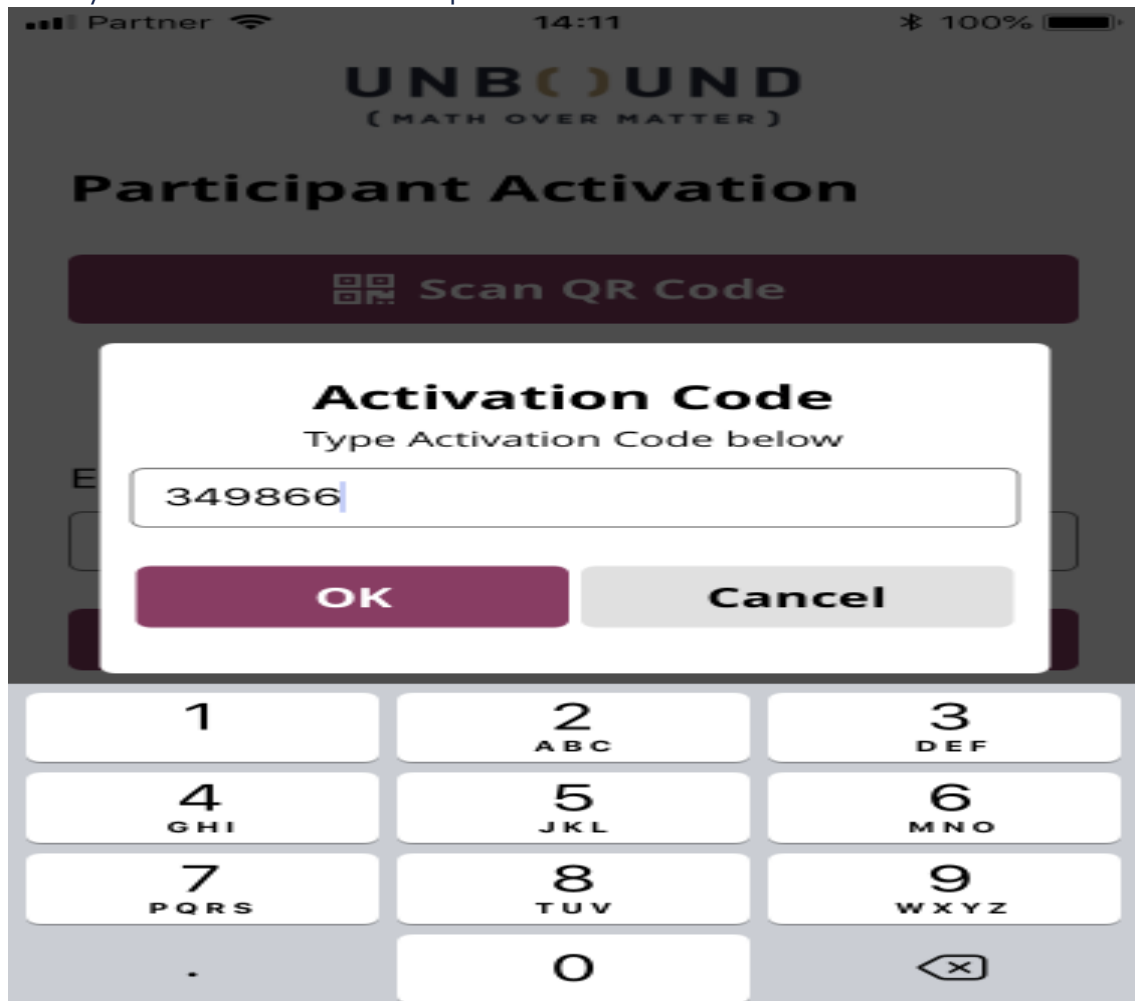
Note

When you tap on the QR code button, you are prompted to let the app access the camera.



2. After the device scans in the Participant ID, tap **Activate**.

3. Enter your activation code and tap OK.

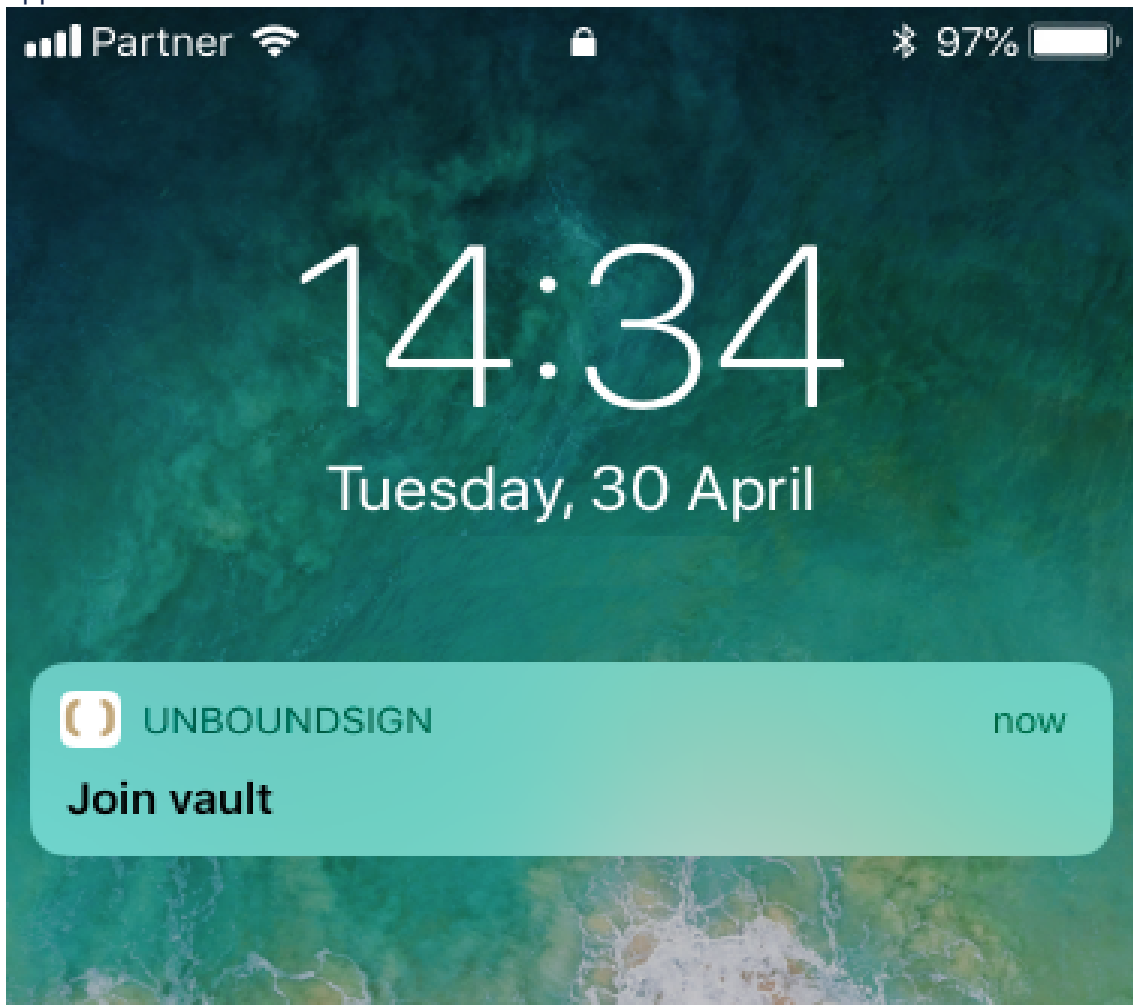


The participant is now activated.

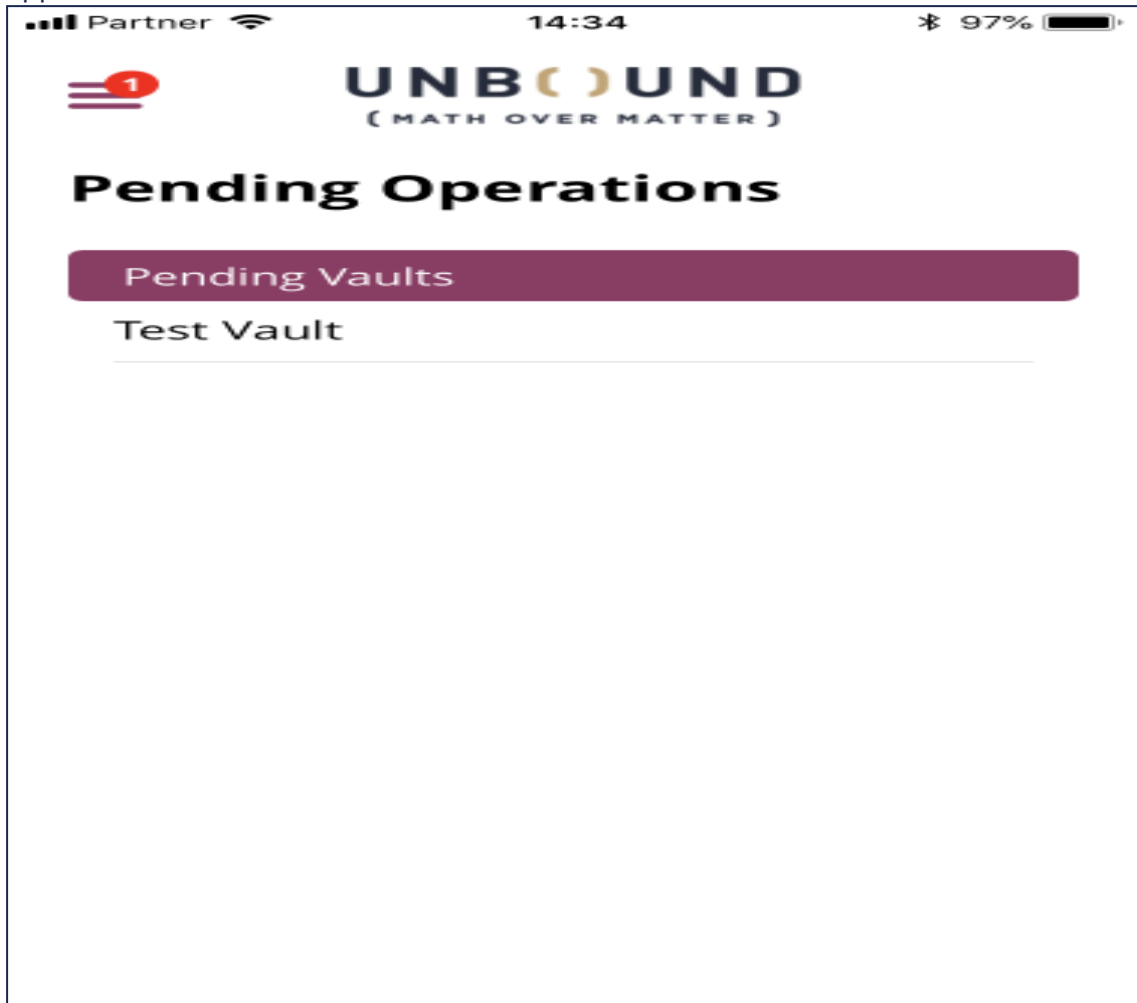
16.4. Vault Approval

A vault is created on the CASP server. Approval requests are then sent to all participants. Approve creation of a vault:

1. When your device receives a request to approve creation of a vault, a notification appears.

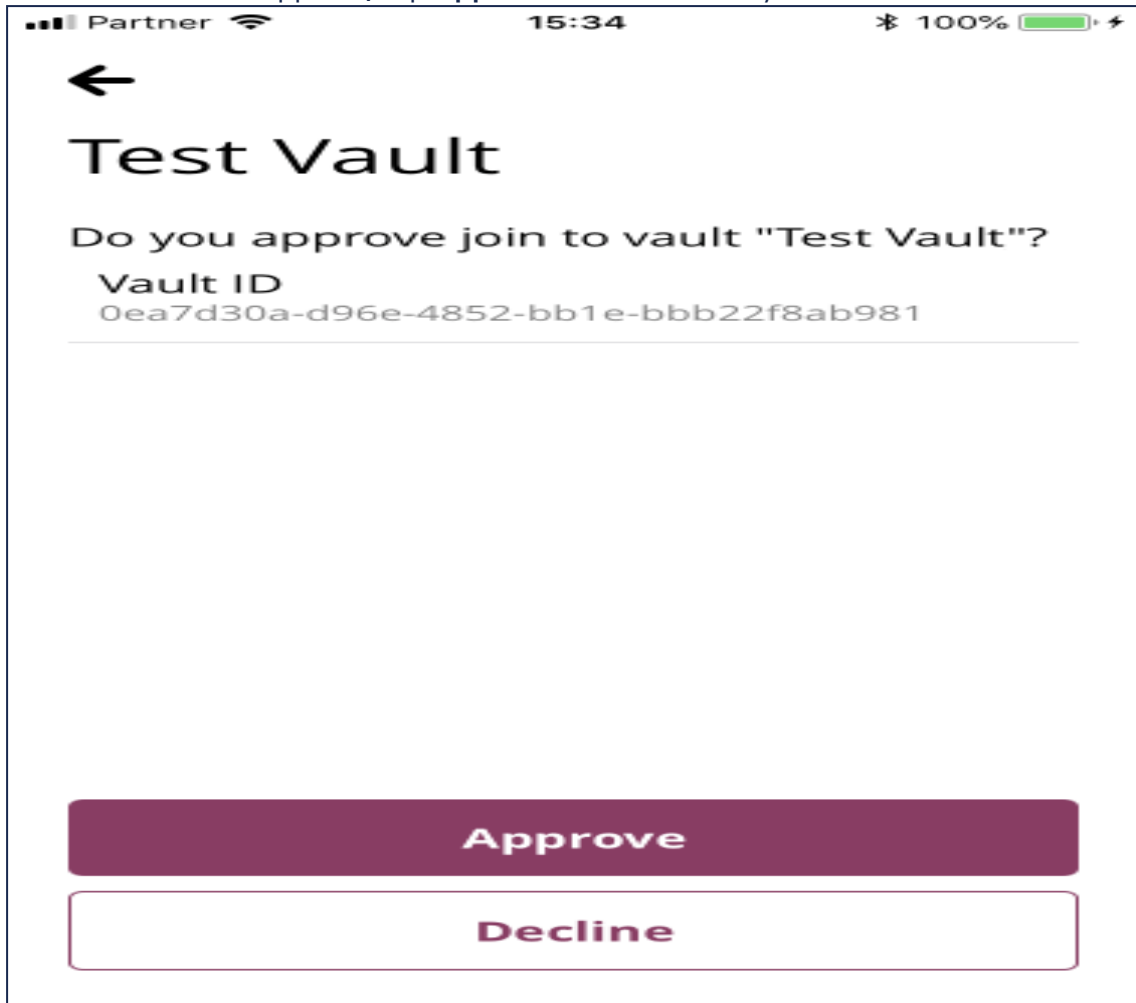


2. Open the app and see that there is a pending vault, called *Test Vault* in the example. Also, see that the menu alerts that there is 1 pending operation (for the vault approval).



3. If you do not see the pending operations, tap the menu button, and then select **Operations**.
4. Tap the vault name.

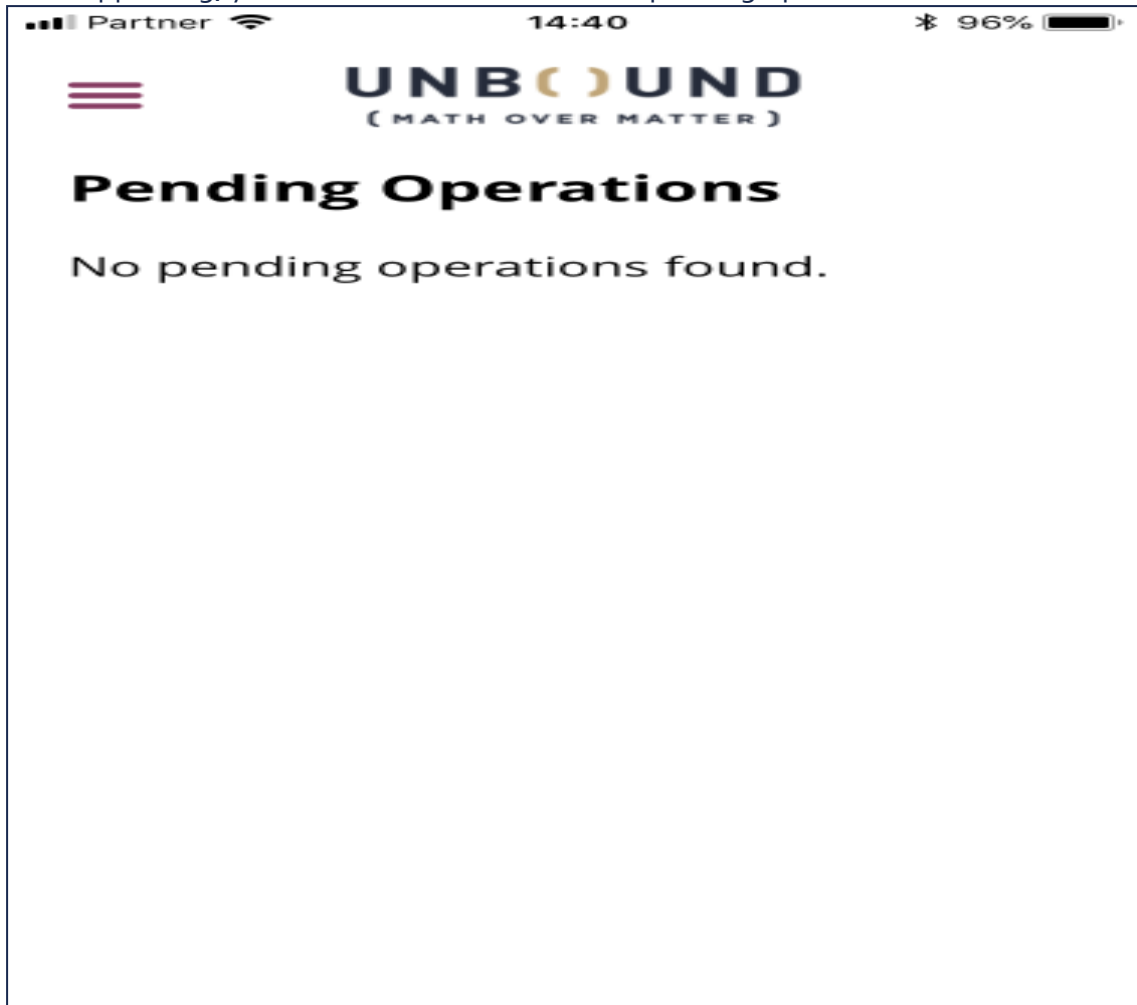
5. On the screen that appears, tap **Approve** and then enter your Face ID (or Touch ID).



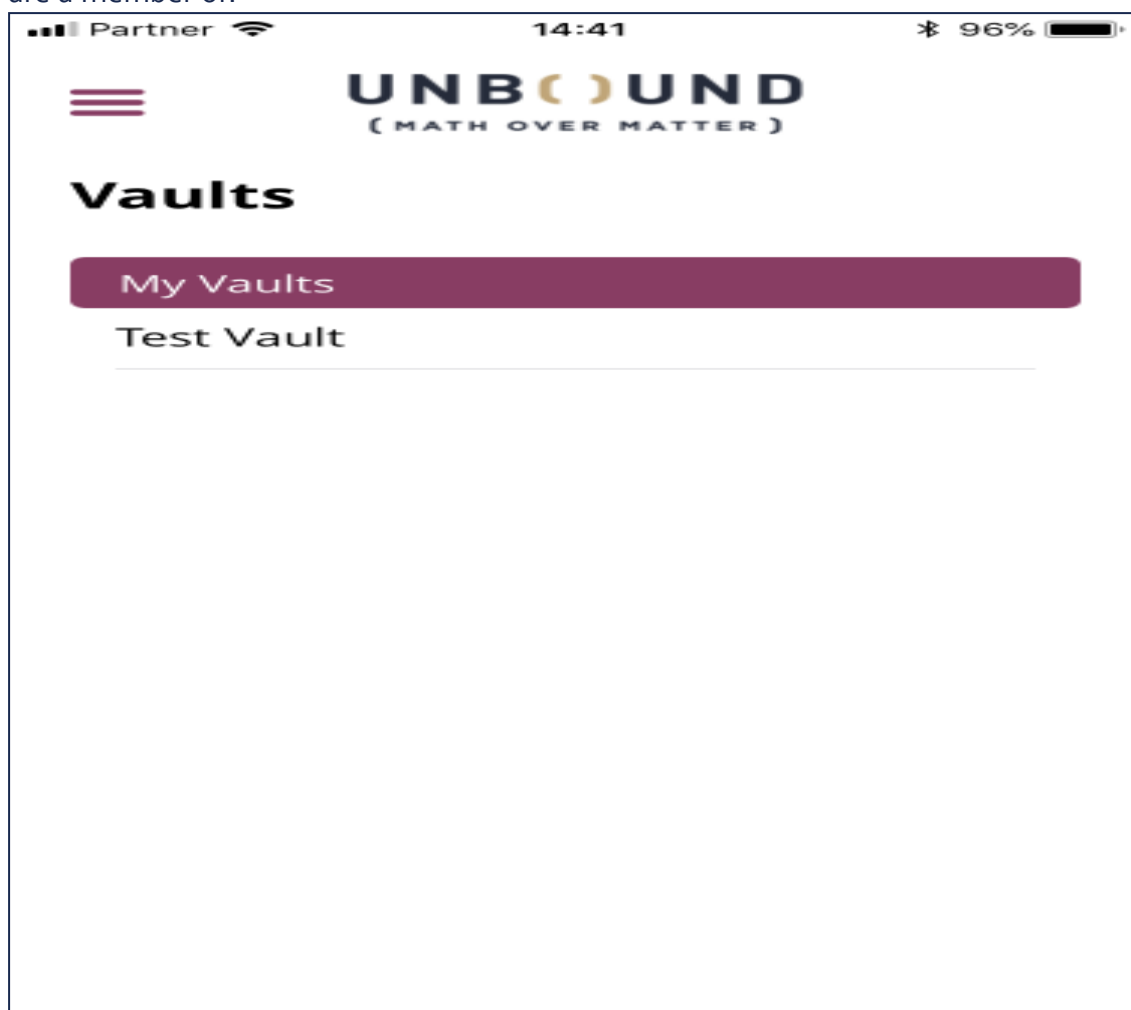
Note

If you are using Face ID, you are prompted to allow Face ID for this app. Click **OK** to allow it.

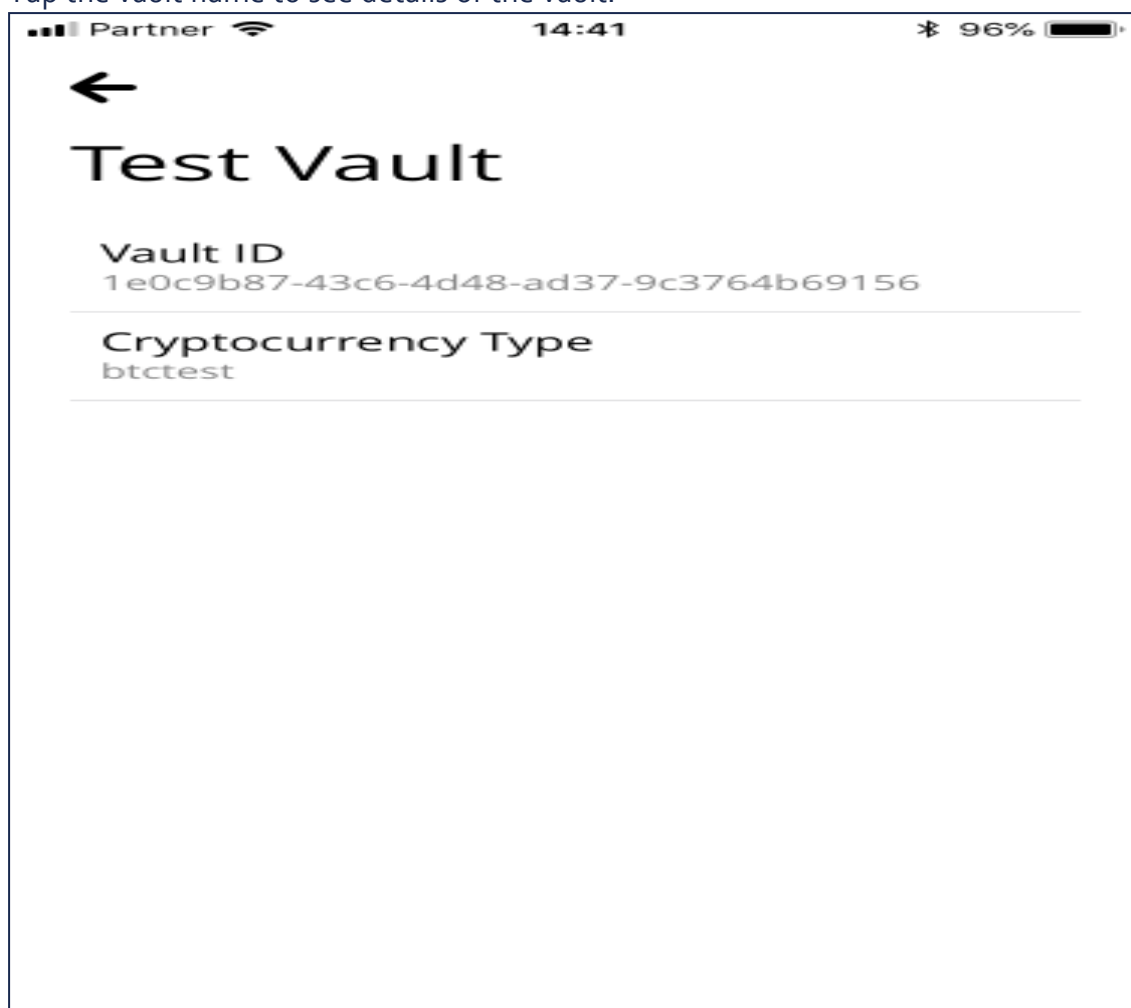
6. After approving, you see that there are no more pending operations.



7. You can tap the menu button and then select **Vaults** to see a list of vaults that you are a member of.



8. Tap the vault name to see details of the vault.



16.5. Operation Approval

When a withdrawal request is made, participants are prompted for approval.

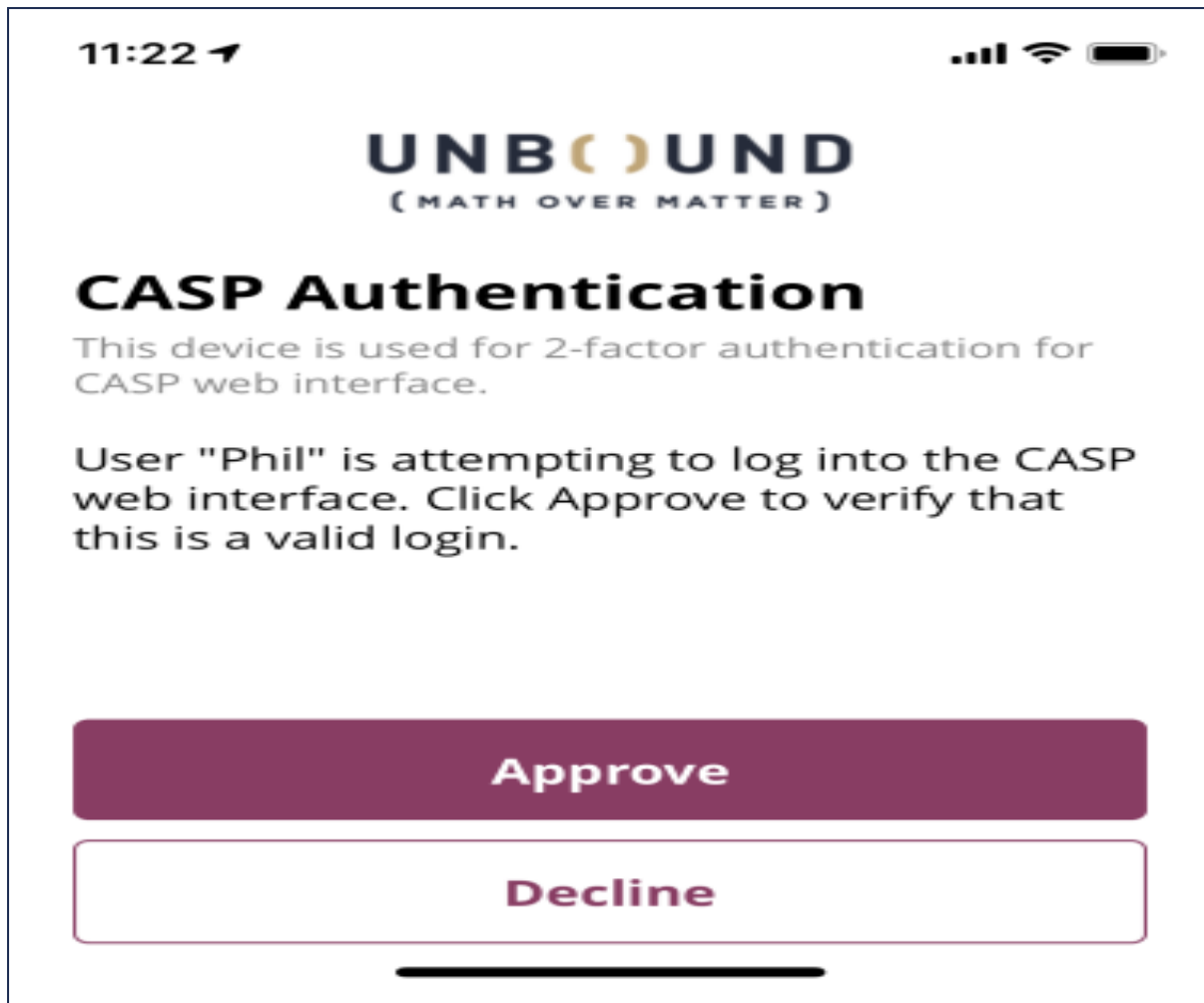
Approve an operation:

1. When your device receives a request to approve an operation, a notification appears.
2. Open the app. The menu alerts you that there is 1 pending operation.
3. Tap the menu, then select **Operations**.
4. On the screen that appears, tap the operation that you want to approve.
5. The details of the operation are shown. Tap **Approve** and then enter your Face ID (or Touch ID).

The operation is now approved.

16.6. 2-Factor Authentication

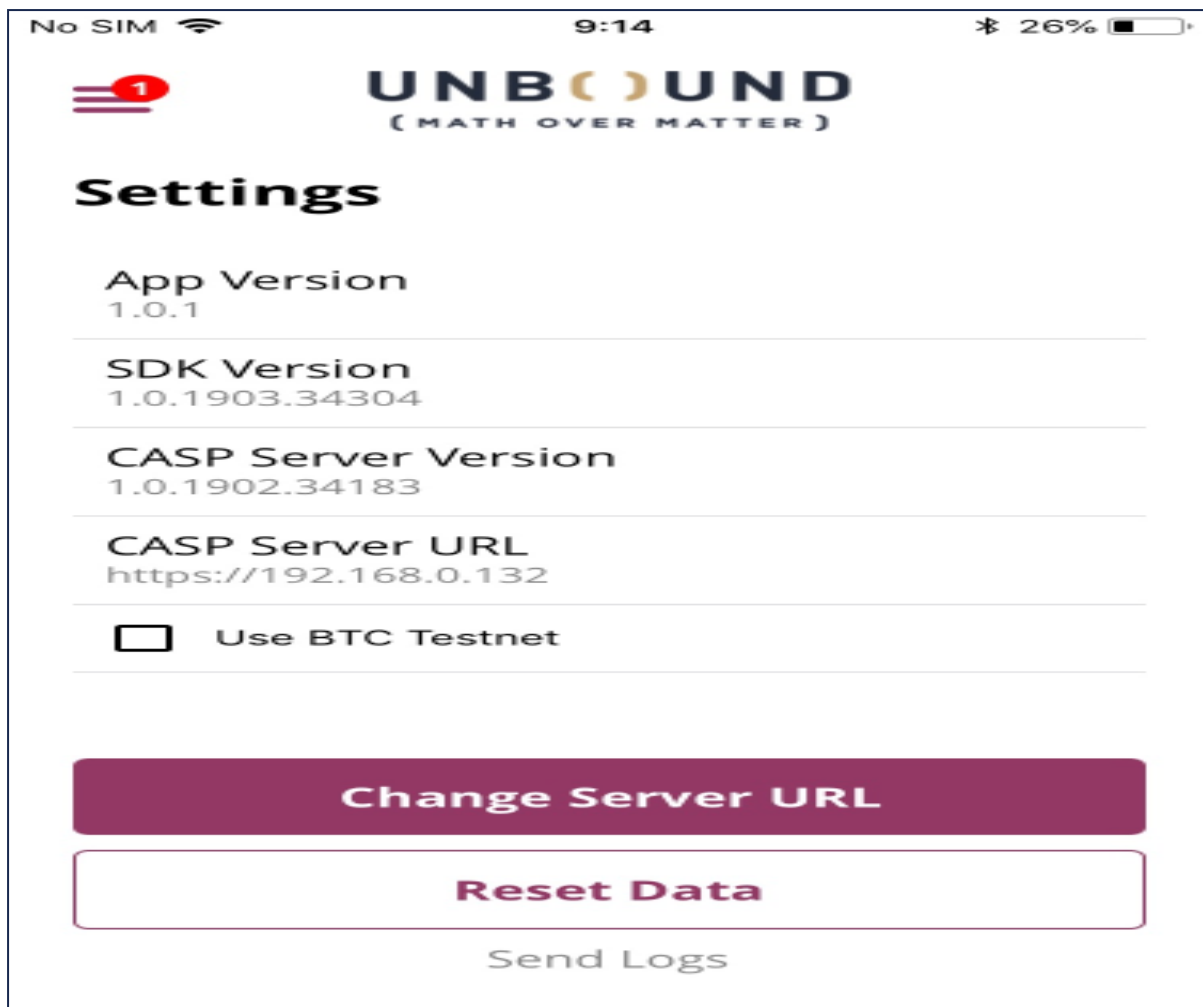
If a user is configured to use 2-factor authentication, a message like the following one is received when that user attempts to log into the CASP web interface.



See [Operators](#) for information.

16.7. Settings

To view the app settings, click on the menu button and then select **Settings**.




The screen provides information about the app, as well as these settings:

- **Use BTC Testnet** - enable this setting if you are using the Bitcoin Testnet.
- **Change Server URL** - click to modify the URL used to access the CASP server.
- **Reset Data** - click to remove all account data from the device. After clicking this button, it displays the initial screen where you enter the server URL.
- **Send Logs** - click to send the app log to Unbound Support, which is only needed if you have an issue with the app.

16.8. Profile

To see your profile, click on the menu button and then select **Profile**.

No SIM 9:15 26%

**UNBOUND**
(MATH OVER MATTER)

My Profile

Name	iphone8
Participant ID	29ce1242-1cec-4a99-ab71-e4cb9652975f
Account ID	052a355b-e936-41a1-8886-6c49124468e9
Account Name	Build49
Email	Iphone8@iphone
Role	dev

17. CASP Bot

A bot is provided that can be used to demonstrate one of the approval mechanisms in your CASP system. This bot can be activated and then can approve pending operations.

Note

The bot must be enabled for listening before it can approve requests.

The CASP bot provides the functions described in the following sections.

17.1. System Requirements

The CASP bot runs on these platforms:

- Linux RHEL/CentOS 7.2 and later
- Ubuntu 16.04

17.2. Installation

Download and set up the CASP bot package.

1. Access the CASP repository from the link provided to you by Unbound.
2. Locate the CASP SDK Package based on the target OS and version.
3. Download the package: *casp-sdk-package.<version>.<os>.tar.gz*.
4. Decompress the package.

The following file is provided for the CASP bot in the *bin* folder:

- *casp-bot-sample.jar*

Note

If you are upgrading the CASP bot and using a directory different from the previous installation, you must copy the key file(s) from the old installation. Key files are found in: `<bot_directory>/bin/<key>.p12`

17.3. Prerequisites

The CASP bot requires one of the following prerequisites:

- Oracle Java JDK 8
- OpenJDK 11. OpenJDK can be obtained from several different sources. The recommended source is [AdoptOpenJDK](#) with the options for *OpenJDK 11* and *HotSpot*.

Note

The bot communicates with CASP over an HTTPS connection. If the bot is not able to connect to CASP due to a certificate issue, add the root ca certificate of the CASP HTTPS certificate to the Java Root CA store using the following command:
`keytool -trustcacerts -keystore <java_path>/lib/security/cacerts -storepass changeit -noprompt -importcert -file chain.pem`

17.4. Activation

Warning

When creating a user for the bot in CASP, make sure that you do not enable 2FA for that user.

To activate the bot participant:

```
java -jar <path_to>/casp-bot-sample.jar -u <https://casp_ip>/casp -p  
<participant_id> -c <activation-code> -w <KeyStore password>
```

Parameters:

- `-jar <path>` - the path to the *casp-bot-sample.jar* file.
- `-c, --activation-code <arg>` - The activation code that your CASP administrator provides.
- `-k, --insecure` - Allow connections without certificate verification.
- `-m, --manual` - Manually approve each operation.
- `-p, --participantID <arg>` - The participant ID.
- `-u, --server-url <URL>` - The URL of the CASP server.
- `-v, --verbose` - Enable verbose logging.
- `-w, --keystorepass <arg>` - Password for the KeyStore.
- `-d` - Java provider name (see [Storing Keys in an External Security Provider](#)).
- `-a` - Java provider password (see [Storing Keys in an External Security Provider](#)).

This activates the bot and stores its share parts in a newly created KeyStore that is protected by the provided KeyStore password.

Warning

If an operator ID is used instead of a participant ID, the bot will not work. See [CASP Operators and Participants](#) for more information about the different types of users in CASP.

17.5. Enable Listening

To start the bot listening to requests:

```
java -jar <path_to>/casp-bot-sample.jar -u <https://casp_ip>/casp -p  
<participant_id> -w <KeyStore password>
```

This loads the bot in listening mode. When a new approval request is received, it is automatically approved.

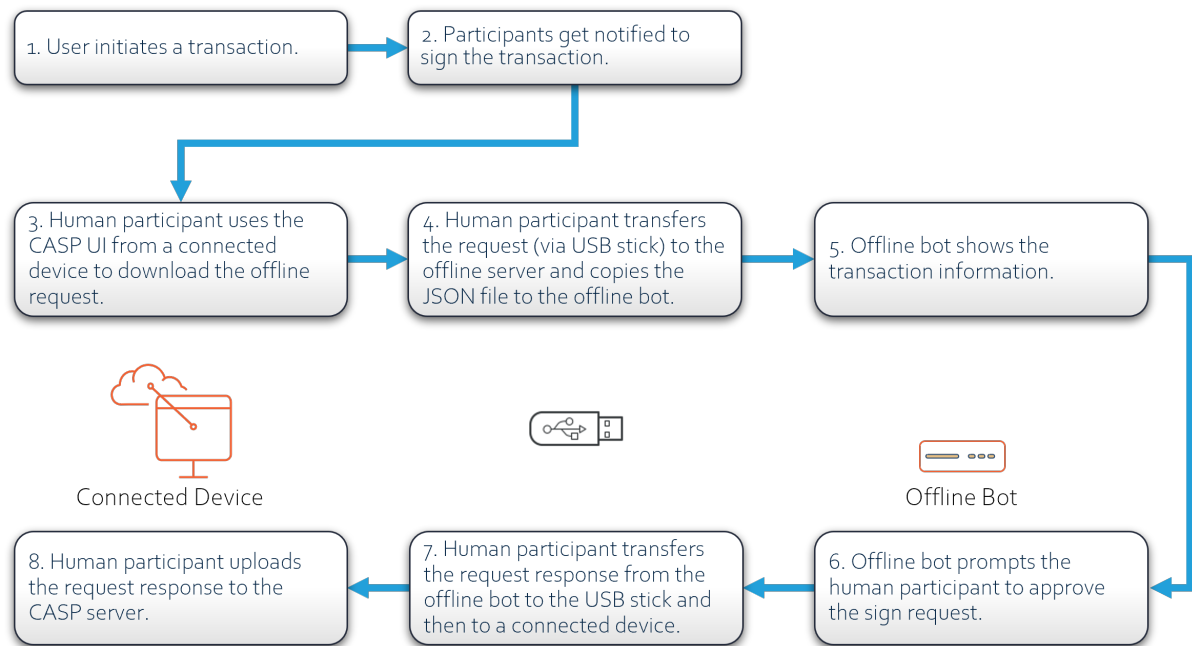
The bot can also run in manual mode, where it asks the user to approve or decline requests. To run in manual mode, add the `-m` flag, as shown in the following command:

```
java -jar <path_to>/casp-bot-sample.jar -u <https://casp_ip>/casp -p  
<participant_id> -w <KeyStore password> -m
```

17.6. Offline Bots

You can create a bot that can be used in an offline mode. This bot does not have any network connection and only sends and receives information when a human participant manually transfers the data using hard media, such as a USB drive.

The following process shows how to create, activate, and manage offline bots. It involves using the [Web Interface](#) or running commands on a terminal that has a network connection to the CASP service, which is called the *base terminal* in the examples, and transferring data to/from the bot device by a human participant. The process is shown in the following figure.



17.6.1. Create and Activate the Offline Bot

1. Create an offline bot.

- If you are using the **Web UI**:
 - a. On the **Users** screen, add a new participant. There is a checkbox to designate that the participant is offline. After clicking **Create participant**, a screen appears with the participant's details and a JSON file is downloaded
 - b. Manually transfer the JSON file to the bot.

- If you are using the **base terminal**:
 - a. Set the *offline* field to *true* when creating the participant.

For example, use this command from the base terminal.

```
curl --request POST \
--url http://localhost:8888/casp/api/v1.0/mng/accounts/a41fcfd-3a4c-4586-a87b-9aea9d876f2f/participants \
--header 'authorization: Bearer dXNlcjExMTE6NjZiOTg4YjgtMjAwMS00MWQxLWF1YzEtMmNkMWNiZGM5NjQ3' \
--header 'content-type: application/json' \
--data '{
  "name": "offline bot-364160696929",
  "email": "302600113578@unboundsecurity.com",
  "role": "offline bot",
  "offline": true
}'
```

Response:

```
{
  "id": "93d4ed76-c3c5-4ff4-a1ae-a2553f472c76",
  "activationCode": "392399",
  "name": "offline bot-481094963642",
  "serverAuthenticationKey":
  "abcd1234abcd1234...abcd1234abcd1234",
  "serverEncryptionKey": "1234abcd1234abcd...1234abcd1234abcd"
}
```

- b. Create a file called *activation_request.json* and put the response into it.
 - c. Manually transfer *activation_request.json* from the base terminal to the bot.
2. On the offline bot, activate the offline participant.

Run the bot with following parameters. See [Activation](#) for information about the activation command.

```
-o -c <activation-code> -a -i <JSON Activation Request File> -t activation_response.json -p <participantID> -w <KeyStore password>
```

Explanation of parameters:

- -o - Offline mode.
- -c, --activation-code <arg> - The activation code that your CASP administrator provides.
- -a - Java provider password
- -i - The input file, created in the previous step.
- -t - The output file for the activation response.
- -p, --participantID <arg> - The participant ID.
- -w, --keystorepass <arg> - Password for the KeyStore.

3. Manually transfer the *activation_response.json* file to the base device.
4. Send the activation response to the CASP server.

- If you are using the **Web UI**:
On the **Users** screen, click the menu icon next to the participant and select *Upload activation data*. Select the file and then click **Upload**. If the incorrect file is used, a JSON error is displayed.

- If you are using the **base terminal**:

For example, from the base terminal use the following command.

```
curl --request POST \  
--url \  
http://localhost:8888/casp/api/v1.0/mng/participants/<participantid>/a  
ctivateOffline \  
--header 'content-type: application/json' \  
--data '  
{ "encActivationRequest": "AAIzRlKk483U0WbPhWMHwV7u9hXON6ET/DC8r...4rLkT  
+xQA==" }'
```

The body of the request should contain the exact contents of the *activation_response.json* file.

The participant should now be active.

17.6.2. Add an Offline Bot to a Vault

Add this participant to one of the quorum groups in the vault.

1. Request server to add bot to vault.
 - If you are using the **Web UI**:
[Add a participant to an existing vault](#)
 - If you are using the **base terminal**:
Use the **Add vault member** (in [Built-in](#) or [BYOW](#)) API.

Place the response in a file called *join_request.json*.

2. Manually transfer the JSON file to the bot and then approve joining the vault on the bot.
Save the response into a file called *join_response.json*.
3. Manually transfer the *join_response.json* file to the base terminal.
4. From the base terminal, send the operation results to the CASP server.

17.6.3. Approve Transactions with the Offline Bot

When executing operations involving an offline bot, the administrator needs to download the operation data and transfer it manually to the offline bot. After the offline bot approves the operations, the admin needs to manually transfer the offline bot response data back to the base terminal and send it to the CASP service.

Note

If there is more than one pending operation for an offline bot, the request data contains them all and the response contains the data for the operations that were approved. This method may save some back and forth between the base terminal and the offline bot.

To approve a transaction (such as create, join vault, and sign):

1. Retrieve the offline participant operations.

- If you are using the **Web UI**:

On the **Users** screen, click the menu icon next to the participant and select *Download offline operation request*. The file downloads.

- If you are using the **base terminal**:

Use this command to retrieve the operations.

```
curl --request GET \
--url
http://localhost:8888/casp/api/v1.0/mng/participants/<participantid>/o
fflineoperations \
--header 'authorization: Bearer
DXN1cjExMTE6NjZiOTg4YjgtMjAwMS00MWQxLWFlYzEtMmNmMWNiZGM5NjQ3'
```

Response:

```
{
  "data":

  "1WqTPSCB8fkxRD7eywADd7hsbu70fo9Rx9qa3J8aiMI4BdRG4ZWSht/9xFBwC6c0jh6X5
SITG1IA==",
  "tag": "nk0h4Yry7meHyywmV/B2aA==",
  "iv": "U66GvIQSEAIq0cRQ",
  "key":
  "OuvWP4CWXHJ2uD1M21g6T...twTCmNeb9w0vPmxjYS+A2KL6JBmHgZ6tas7kI2sgk8g=
=",
  "participantId": "<participantid>",
  "sig":
  "0+g/+MK/oB1DQR7a/nS/41ioJGbQ8YTbUQg2rvc9FABp7vde0qEVbNbpEVKzrq+hYWw=
="
}
```

Place the response in a file called *operations_request.json*.

2. Manually transfer the JSON file to the bot.

3. Run the bot with following parameters. See [Activation](#) for information about the activation command.

```
-o -q operations_request.json -n operations_response.json -p <participantid>
-w 12345678 -m
```

Parameter descriptions:

- -o - offline mode
- -q - the file from the previous step containing the operations.
- -n - an output file for the response.
- -p, --participantID <arg> - The participant ID.
- -w, --keystorepass <arg> - Password for the KeyStore.
- -m - manual approval mode.

In addition to the standard information that the bot displays, when working offline the bot prints out transaction information, including the hash to sign, the raw transaction, and which public keys are used. The bot then prompts you to approve or deny the operation.

The following image shows an example of a sign operation.

```
ubuntu@ip-192-168-0-237:/tmp$ java -Djava.library.path=/tmp/bin/ -jar /tmp/bin/BotSigner.jar -p $ID -w 12345678 -o -i ./a7.json -t ./a8.json -m
Manual approval mode
offline participant mode
Bot init was successful
Raw transactions:
eb80843b9aca0082520894a130dcc043bf348d67985a61a8d14ea5a4678743880163325eebfb00080038080
Data to sign:
901da89a75d3d51b8dfa9343e5b34fe073270152d4bcc61378f6ca80a0542f30
Public keys:
3056301006072A8648CE3D020106052B8104000A03420004A5A8162DAD2014323224F6D0870996287AC74BA1D606D18EBD9308531D9ECD3B869C1717F6317281E47EB4E4BA7424C

Do you approve the following operation?
Kind      : QUORUM_SIGN
Operation ID : fe526e93-624e-4831-a440-060756f033bf
Description : 💰 money money money money
Vault ID   : 0fca894e-c062-4911-9939-0c4545064b69

1. No
2. Yes
3. Always approve
2
successfully approved offline participant operations
```

4. Manually transfer the *operations_response.json* file to the base terminal.
5. Send the operation results to the CASP server.
 - If you are using the **Web UI**:
On the **Users** screen, click the menu icon next to the participant and select *Upload offline operation response*. Select the file and then click **Upload**. If the incorrect file is used, a JSON error is displayed.
 - If you are using the **base terminal**:
Send the operation results to the CASP server.

```
curl --request POST \
--url
http://localhost:8888/casp/api/v1.0/mng/participants/<participantid>/offlineoperations \
--header 'authorization: Bearer dXNlcjExMTE6NjZiOTg4YjgtMjAwMS00MWQxLWFLYzEtMmNmMWNiZGM5NjQ3' \
--header 'content-type: application/json' \
--data '{
  "data":
    "AANDs8Wf+1+oCwsjS3mIejpzHxUXqUFqY8/YYHdf+CAiadrS4eud/mPzNJJ0yhunQ3opS
    Xy=",
  "sig":
    "MEYCIQD83JSMDxjdaxG/UCR1ojPu5hTtravJPtrwWcyCUoFz/wIhAMpe2seX/91Bmxkpg
    2fgJ"
}'
```

The body of the request should contain the exact contents of the *operations_response.json* file (created in the previous step).

17.7. Storing Keys in an External Security Provider

This section describes how to use the CASP bot to store encryption and signing keys in an external security provider, such as an HSM or a smartcard. This solution uses the PKCS#11 standard through a Java interface that is used to communicate between the CASP bot and the external security provider.

Implement the solution using this procedure:

1. Set up the Java security provider, using either one of the following:
 - a. If you have a Java security provider, then locate the Java provider name and password. They must be specified when starting the CASP bot.
 - b. If your security provider has a PKCS#11 library (an .so file), then configure it. Point the Sun PKCS#11 (that comes built-in with Java) to this provider. Provide the Sun PKCS#11 provider name and password that is configured to work with PKCS#11.
2. Start the CASP bot using the options to specify external key storage name and password.

For example:

```
java -jar <path_to>/casp-bot-sample.jar \  
-u <https://casp_ip>/casp \  
-p <participant_id> \  
-c <activation_code> \  
-w <KeyStore password> \  
-d <Java provider name> \  
-a <Java provider password>
```

18. Troubleshooting

If you have a problem with either the CASP server or a device running CASP, use the following steps to help troubleshoot the issue.

- Check that the CASP server and other components are healthy.
Run the checks described in [Testing the CASP System](#).
- Follow the instructions in [Participant replaces a phone](#) if any of these situations occur:
 - The app is removed from the device.
 - The user resets the app data.
 - The user gets a new phone.
- **For the mobile app:**
 - Check that the URL of the CASP server is correct.
See [Settings](#) in the [Mobile App](#).
 - Check that the internet connection from the mobile device is working.
 - Check that the mobile device can access the server. You may need to check that the device has a valid certificate for the server.
 - If the user removes the pin **and** biometric authentication, no operations will be able to be approved. The user must set a new pin or biometric authentication.
- If a participant was deactivated and then activated, the participant will not be able to approve any operations until they are re-added to all relevant vaults.
- **For Built-in Wallets:**
 - For both Bitcoin and Bitcoin Testnet, when retrieving the balance on a BIP44 vault with sub-accounts, you may encounter a situation where the Blockset BTC/BTCTEST token limit is reached. It returns an error code such as:
`API calls limits have been reached.`

The following parameters can be specified in the configuration file, *production.yaml*, to control the behavior of Blockset requests.

Blockset:

`maxConcurrentRequests: 10 // # of requests allowed to be sent at once to Blockset`

`minRequestTimeMs: 100 // The minimum time between one request and another`

`maxRetries: 10 // # of times to retry once a request to Blockset fails with 429 - too many requests`

- Vault creation:
 - You cannot generate a new activation code for a user if there is a pending request to join a vault. This means that if you create a new vault and have a participant that lost his phone before approving to join the vault, you need to cancel the vault creation operation.

18.1. CASP Restore SO Password

The password for the CASP **so** user may become locked if the incorrect password is used a number of times. In such a case, you can reset the password with this script.

Syntax:

```
casp_restore_so \  
--conf-directory=<location_of_files>
```

The script has this parameter:

Parameter	Required	Description
conf-directory	optional	Location of the <i>casp.conf</i> and CORE certificate files.

The response on success is:

```
Restore so user was successful
```

18.2. Contact Support

If you are still having the issue, [contact Unbound Support](#).

19. Production Checklist

The following sections provide information about testing your deployment before moving it to production. It contains test scenarios as well as questions to answer prior to release.

19.1. Installation

1. Do you need a high availability installation? If so, refer to [High Availability](#).
2. Did you segregate the CORE EP and Partner servers?
 - Does each server have different credentials/SSH keys? This mitigates an attack from compromised credentials.
 - Each server should have a different security context, separate VMs or containers as a minimum. Deployment architectures spanning across different environments (such as on-premises and cloud, across different cloud service providers) are easy to create and maintain, providing a high level of security due to an excellent level of segregation.
3. Did you use 2-factor authentication for the cloud account?
4. Did you use Unbound's OS hardening best practices for any local CORE server? If you need this information, [contact Unbound Support](#).
5. Did you restrict access to the servers using firewalls and/or security groups? See [Server Requirements](#) for the list of necessary ports.
6. The CORE root CA and EP certificates need to be periodically updated. See [PKI Scripts](#) for information on how to set up this renewal.

19.2. Vaults

Your quorum groups should have more participants than the required minimum number for approval. This setup allows replacing a participant, which may be necessary if a [Participant replaces a phone \(reactivate\)](#) or a [Participant put on hold globally or in a specific vault \(suspend\)](#).

For example, your group has 3 participants required for approvals, and it has 4 participants. A participant loses his phone, and the remaining 3 can reactivate the participant on a new phone.

1. Do all your vault groups have enough members?
2. Did you consider having vault members that are bots that can be stored securely in cold storage to be used in emergency cases?
3. How resilient is your vault?
 - Calculate: (# of members) + (# of bots) – (minimum required for approval)
 - This number tells you how many members your vault can tolerate becoming unavailable (such as a lost phone).

19.3. Backup and restore

1. Did you back up the configuration files for the CASP Orchestrator servers?
2. CORE-level backup and restore.
 - Backup details for CORE can be found in [Backup and Restore Overview](#).
 - Add automation to periodically run the backup process.
 - Ideally, you should backup after every new vault is generated or preferably for every sub-account.
3. In parallel to backing up CASP components, you should backup vaults, as described in [Key Backup and Restore](#), outside of at the CASP system, allowing you to restore to:
 - A standalone compatible wallet.
 - A custodian system where the customer wants to be able to recover funds even if the custodian service ceases to operate.
 - Moving from CASP to a different vault system.
4. Is the encrypted backup data stored in the CASP database or in a different location?
5. Is the restore key in a safe location outside of the custodian system?

Note

It is recommended to put the private key in cold, disconnected storage. The key can also be duplicated for resiliency.

Note

Ensure that the private key was never stored on any non-secure system or that it was securely deleted. If you delete the private key from a hard drive it can still be retrieved.

6. Did you test the following backup scenarios?
 - A vault member loses their phone and needs to be activated on their new phone.
 - A vault member needs to be replaced.
 - CORE becomes unavailable.
 - CASP server becomes unavailable.
 - CASP database becomes unavailable.
 - Total system loss, including CASP, CORE, and the database.

19.4. Security

1. Do you have any secret information in the *casp.conf* file?
 - You can encrypt the passwords contained in this file with the [CASP Setup Utility](#).

19.5. Bot Security

1. Did you use different credentials to access the system where the bot is stored?

Appendix A. PostgreSQL

A.1 Install PostgreSQL

Note

In the examples, we refer to PostgreSQL-9.6. In the subsequent commands, you should replace the version number with the version that you are using.

Installation of PostgreSQL requires the following “yum install” steps.

```
sudo yum -y localinstall http://yum.postgresql.org/<Postgresql version>.rpm
sudo yum install postgresql96-server
sudo yum install postgresql-contrib
```

A.2 Start PostgreSQL as a Service

Start PostgreSQL per your Linux Platform (RHEL 7+ and CentOS):

```
sudo /usr/pgsql-9.6/bin/postgresql96-setup initdb
sudo systemctl start postgresql-9.6.service
```

Verify that PostgreSQL service shall start automatically when the OS (re)starts:

```
sudo systemctl enable postgresql-9.6.service
```

A.3 Tweak PostgreSQL Permissions

1. Edit `/var/lib/pgsql/9.6/data/pg_hba.conf` as follows:

- Set all METHOD(s) to trust.
- Add a new entry to the table.

host	all	all	0.0.0.0/0	trust
------	-----	-----	-----------	-------

The file should show the following entries:

# TYPE	DATABASE	USER	ADDRESS	METHOD
# "local" is for Unix domain socket connections only				
local	all	all		trust
# IPv4 local connections:				
host	all	all	127.0.0.1/32	trust
# IPv6 local connections:				
host	all	all	:::1/128	trust
host	all	all	0.0.0.0/0	trust

2. Edit `/var/lib/pgsql/9.6/data/postgresql.conf` as follows:

Set the `listen_address` as follows:

```
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
listen_addresses = '*'          # what IP address(es) to listen on;
                                # (change requires restart)
port = 5432                     # (change requires restart)
```

Appendix B. Web UI Manual Installation

The CASP Web UI is installed automatically with the [CASP Package Installation](#). These instructions are provided in case you need to manually install the Web UI.

Do the following on your server to install the web interface:

1. Download and set up the **CASP User Interface** package.
 - a. Access the CASP repository from the link provided to you by Unbound.
 - b. Select your platform and version and then download the corresponding *casp-ui-{version}.tar.gz* file.
 - c. Decompress the package.

2. Deploy CASP UI.

```
sudo cp -r caspui /opt/casp/tomcat/webapps
```

3. Set up Apache.
4. Install Apache http daemon.

```
sudo yum -y install httpd
```

5. Configure Apache to start on boot.

```
sudo systemctl start httpd  
sudo systemctl enable httpd
```

6. Modify *httpd.conf*.

```
sudo nano /etc/httpd/conf/httpd.conf
```


7. Add the following settings.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so

<VirtualHost *:80>
    ProxyPreserveHost On

    #CASP
    ProxyPass /casp http://localhost:8080/casp
    ProxyPassReverse /casp http://localhost:8080/casp
    ProxyPass /caspi http://localhost:8080/caspi
    ProxyPassReverse /caspi http://localhost:8080/caspi

    #BTC
    ProxyPass /btc http://localhost:8080/btc
    ProxyPassReverse /btc http://localhost:8080/btc
    ProxyPass /btctest http://localhost:8080/btctest
    ProxyPassReverse /btctest http://localhost:8080/btctest

    #ETH
    ProxyPass /eth http://localhost:3000/eth
    ProxyPassReverse /eth http://localhost:3000/eth
    ProxyPass /ethctest http://localhost:3001/ethctest
    ProxyPassReverse /ethctest http://localhost:3001/ethctest
</VirtualHost>
```

8. Allow access to localhost.

```
sudo /usr/sbin/setsebool -P httpd_can_network_connect 1
```

9. Restart Apache.

```
sudo systemctl restart httpd
```

10. Test the Apache configuration.

11. Test CASP.

```
curl http://localhost/casp/api/v1.0/mng/status?withDetails=true
```

12. Test BTC Provider.

```
curl http://localhost/btc/api/v1.0/status?withDetails=true
```

13. Test BTCTEST Provider.

```
curl http://localhost/btctest/api/v1.0/status?withDetails=true
```

14. Test ETH Provider.

```
curl http://localhost/eth/api/v1.0/status?withDetails=true
```

15. Test ETHTEST Provider.

```
curl http://localhost/ethtest/api/v1.0/status?withDetails=true
```

Note

All responses should be the same as in previous sections.

Appendix C. Using Databases Over SSL

Unbound CASP can use PostgreSQL or MySQL (installed via RDS) over SSL. The following instructions explain how to set up the environment.

C.1 RDS PostgreSQL Over SSL

C.1.1 Step 1: Create an RDS parameter group

1. In the RDS console, create new parameter groups.
 - a. In the RDS console, click **Parameter groups**.
 - b. Click **Create parameter group**.
 - c. In the screen that opens, enter a group name and description. The *Parameter group family* should be set to *postgres9.6*.
 - d. Click **Create**.
2. Click the name of your parameter group.
3. Click **Edit parameters**.
4. Under *Parameters*, search for *rds.force_ssl*.
5. Change this parameter value to 1.
6. Click **Save changes**.

C.1.2 Step 2: Update your RDS instance

1. Find your RDS instance, then click on the name.
2. Click **Modify**.
3. In the **Database options** section, change the *DB parameter group* to the one created above.
4. Click **Continue** at the bottom of the screen.
5. Reboot the instance by clicking **Actions**, then **Reboot**.

C.1.3 Step 3: Configure PostgreSQL

Update the PostgreSQL configuration. See the Amazon documentation for more information.

1. Log into PostgreSQL on your RDS server.

For example:

```
psql -h server.us-east-1.rds.amazonaws.com -p 5432 -U sa -d casp
```

2. Update the configuration by running each of the following commands.

```
create extension sslinfo;
select ssl_is_used();
select ssl_cipher();
```

3. Exit the RDS server.

C.1.4 Step 4: Configure SSL

1. Try and log into your RDS server.

```
psql -h server.us-east-1.rds.amazonaws.com -p 5432 -U sa -d casp
```

You get an error message like this one:

```
psql: FATAL: no pg_hba.conf entry for host "18.208.181.219", user "sa",
database "casp", SSL off
```

2. Download the certificate:

```
wget https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem
```

3. Copy the certificate into the PostgreSQL folder.

```
sudo cp rds-combined-ca-bundle.pem /root/.postgresql/root.crt
```

C.1.5 Step 5: Configure CASP

Update the [CASP configuration file](#).

1. In *casp.conf*, locate the PostgreSQL database URL line.

For example:

```
database.url=jdbc:postgresql://localhost:5432/casp
```

2. Add "?ssl=true" to the URL.

For example:

```
database.url=jdbc:postgresql://localhost:5432/casp?ssl=true
```

3. Restart CASP.

```
sudo service casp.tomcat restart
```

C.1.6 Step 6: Check the SSL connection

Log into the RDS server.

```
psql -h server.us-east-1.rds.amazonaws.com -p 5432 -U sa -d casp
```

The response looks like:

```
psql (11.2, server 9.6.3)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits:
256, compression: off)
```

```
Type "help" for help.  
casp=>
```

You are now successfully logged into PostgreSQL on your RDS server using an SSL connection.

C.2 RDS MySQL Over SSL

Unbound CASP can use MySQL (installed via RDS) over SSL. The following instructions explain how to set up the environment.

C.2.1 Step 1: Configure the certificate

1. Download the certificate.

```
wget https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem
```

2. Convert the certificate format from *pem* to *der*.

```
openssl x509 -outform der -in rds-combined-ca-bundle.pem -out rds-combined-ca-bundle.der
```

3. Add the certificate to the *security/cacerts* folder for Java via *keytool*.

```
sudo keytool -import -alias rds-combined-ca-bundle -keystore <JAVA  
HOME>/lib/security/cacerts -file rds-combined-ca-bundle.der  
Enter keystore password: <Keystore Password>  
Trust this certificate? [no]: yes
```

C.2.2 Step 2: Configure CASP

Update the [CASP configuration file](#).

1. In *casp.conf*, locate the MySQL database URL line.

For example:

```
database.url=jdbc:mysql://localhost:3306/casp
```

2. Add the SSL flags to the URL.

For example:

```
database.url=jdbc:mysql://<SERVER NAME>:3306/casp?useSSL=true
```

If you have an in-house CA and/or domain verification is required:

```
database.url=jdbc:mysql://<SERVER  
NAME>:3306/casp?useSSL=true&verifyServerCertificate=true&trustCertificateKey  
StorePassword=<Keystore Password>
```

3. Restart CASP.

```
sudo service casp.tomcat restart
```

C.2.3 Step 3: Check the SSL connection

Log into the RDS server.

```
mysql -h <SERVER NAME> -p5432 -u sa casp --ssl-ca=rds-combined-ca-bundle.pem --ssl-mode=VERIFY_IDENTITY
```

After logging in, execute `\s` to view the connection parameters. Verify that the `SSL` field shows that a cipher is in use.

The response looks like:

```
MySQL [casp]> \s
-----
mysql Ver 15.1 Distrib 10.2.18-MariaDB, for Linux (x86_64) using readline 5.1
Connection id:          667
Current database:       casp
Current user:           sa@31.154.180.82
SSL:                    Cipher in use is AES256-SHA
Current pager:          stdout
Using outfile:           ''
Using delimiter:        ;
Server:                 MySQL
Server version:         8.0.15 Source distribution
Protocol version:       10
Connection:             servername.rds.amazonaws.co via TCP/IP
Server characterset:    utf8mb4
Db characterset:        utf8mb4
Client characterset:    utf8
Conn. characterset:     utf8
TCP port:               3306
Uptime:                 2 hours 8 min 6 sec
Threads: 3 Questions: 6587 Slow queries: 0 Opens: 1078 Flush tables: 2 Open
tables: 488 Queries per second avg: 0.857
```

You are now successfully logged into MySQL on your RDS server using an SSL connection.

Appendix D. Installing MySQL

These instructions describe how to install MySQL locally, which may be done for a POC, UAT, or other testing environment.

1. Install MySQL.

You can get the repository from: <https://dev.mysql.com/downloads/repo/yum/>.
For example:

```
sudo yum install https://dev.mysql.com/get/mysql57-community-release-el7-9.noarch.rpm
sudo yum install mysql-server
```

2. Start MySQL.

```
sudo systemctl start mysqld
sudo systemctl status mysqld
```

During the installation process, a temporary password is generated for the MySQL root user. Locate it in the *mysqld.log* with this command:

```
sudo grep 'temporary password' /var/log/mysqld.log
```

3. Configure MySQL.

```
sudo mysql_secure_installation
```

This command prompts for the default root password and then requires you to change it.

4. Test MySQL.

```
mysqladmin -u root -p version
```

5. Create the database for CASP.

a. Log into MySQL.

```
mysql -p<root password> -u root
```

b. Create the database called *casp*.

```
create database casp
```

6. Load the CASP schema.

```
mysql -p<root password> -u root casp < /opt/casp/sql/casp-mysql.sql
```

Appendix E. Create a CSR for a CASP Vault

Use this procedure to get a signed CSR for a key stored in CASP.

1. Create a secp256r1 deterministic quorum vault.
2. Get the vault's public key.
3. Convert the public key into binary format and store it in a file called *ec_public_key.der*.

You can use the script *hex_to_bin.py* provided by Unbound for this conversion.

4. Convert the *ec_public_key.der* file into *pem* format using the following command:

```
openssl ec -in ec_public_key.der -inform DER -out ec_public_key.pem -outform PEM -pubout -pubin
```

5. Create the *ec_hash.bin* file with the following command:

```
casp_csr_tool --prepare ec_hash.bin --pubkey ec_public_key.pem --hash sha256 --subject "CN=test"
```

6. Convert the *ec_hash.bin* content into hex.

You can use the script *bin_to_hex.py* provided by Unbound for this conversion.

7. Use the endpoint *Sign a transaction* in the BYOW vault with the previous step's output as *dataToSign*.
8. Get sign operation and copy the signatures property part.
9. Convert the previous step's output into binary and store it in a file called *ec_signature.bin*.
10. Create *ec_csr.pem* with the following command:

```
casp_csr_tool --output ec_csr.pem --format pem --pubkey ec_public_key.pem --signature ec_signature.bin --hash sha256 --subject "CN=test"
```

For example:

```
[ubuntu@casp ~]$ ./casp_csr_tool --output ec_csr.pem --format pem --pubkey ec_public_key.pem --signature ec_signature.bin --hash sha256 --subject "CN=test"
```

11. Use the following command:

```
openssl req -text -noout -verify -in ec_csr.pem
```