BLOCKCHAIN PRESENCE

A Low-Cost Smart Contract Oracle with On-Chain Authentication

Christian Ewerhart[1]

This version: May 8, 2022

Abstract. *Blockchain Presence* is a secure and low-cost platform for smart contract information. Any interested individual, corporation, or public authority may become a provider of data. An innovative three-step on-chain authentication method ensures that the information delivered to the smart contract may be trusted as coming from the selected sender. The leanness of the decentralized protocol contrasts with existing smart-contract oracles that often rely on unauthorized API queries, off-chain authentication, and inefficient collateral tokens. The level of security achieved by the protocol opens the way for entirely new use cases.

[1] Department of Economics, University of Zurich, Schönberggasse 1, 8001 Zürich, Switzerland, email: christian.ewerhart@econ.uzh.ch

Contents

1. Background

1.1 Smart contracts

Smart contracts (Szabo, 1997) are an innovative way to foster cooperation between parties that do not necessarily trust each other. Since neither courts nor lawyers are involved, smart contracts are often considered a substitute for legal agreements and third-party intermediation. The implementation of smart contracts relies on distributed ledger technologies. That is, the state of the cooperative process is stored in a redundant way on a large number of computer nodes with access to the internet, and a decentralized consensus protocol ensures that the information represented in the individual nodes is consistent. Once deployed, smart contracts are executed via function calls, and typically cannot be modified.

One example of a smart contract is flight-delay insurance, in which an anonymous insurer promises to compensate an equally anonymous insured if a specific flight is delayed (Raskin, 2017). Further use cases arise in electronic identification, decentralized finance, logistics, and prediction markets.

So far, most smart contracts have been realized on the Ethereum platform (Buterin, 2014). The most popular programming language employed on the Ethereum platform is Solidity (Antonopoulos and Wood, 2018).

1.2 Blockchain oracles

Most smart contracts require external information. This input can take many forms. In the flight-delay-insurance example above, the smart contract must be informed whether the flight arrived on time or was delayed. Because financial interests are involved, however, it is not straightforward to ensure that the contract receives correct information. In other words, blockchains face a problem analogous to the internet's fake-news problem.

Oracles are information systems that provide external data to smart contracts in a reliable way. They may therefore be understood as mechanisms that bridge the gap between the off-chain and on-chain worlds. Allowing smart-contract conditions to depend on real-world data broadens their scope dramatically. For this reason, oracles are widely regarded as an important area of technological development in the blockchain ecosystem.

For developers using a smart-contract oracle, two quality dimensions matter most.

- **Reliability.** A data item requested by a smart contract must be delivered on time, in the correct format, and with the correct value. If the information is unavailable for unforeseen reasons, this must also be communicated to the smart contract.

- **Authenticity.** The requested information must verifiably originate from the selected source. The smart contract must be able to condition its execution on whether the data came from that source.

Most existing oracles seek to provide both reliability and authenticity. Reliability, however, is often either underemphasized or implemented through extensive multi-sourcing and costly collateral tokens. Authenticity also remains difficult. Internet protocols and trusted hardware environments can support data authenticity, but they do not by themselves allow a smart contract to verify that the data came from the selected original source.

1.3 Blockchain Presence (BCP)

The Blockchain Presence protocol advances the state of the art in two substantial ways. For reliability, it uses commitments that require information providers to make public promises recorded on the blockchain. For authenticity, it introduces on-chain authentication. This mechanism is designed to ensure that data originates from the selected source and from no other source.

Thus, the key difference between *Blockchain Presence* and existing smart contract oracles is the commitment and on-chain authentication of the information providers. In the version implemented on the Ethereum platform, these features are combined with a lean transmission and payment protocol. We therefore believe that Blockchain Presence is well-positioned to ultimately become one of the key players in the future market for smart contract oracles.

2. Mainnet Modules

2.1 Overview

This section describes the Solidity implementation of the *Blockchain Presence* mainnet modules.

Blockchain Presence is a decentralized application (DApp). It consists of the following components:

- Website
- BCP Mainnet Modules
- BCP Testnet Modules
- Sender Convenience Applet ("SCA")
- Use Case Library

The mainnet and testnet modules have been implemented via Solidity on the Ethereum platform. The use case library is likewise written in Solidity.

The mainnet contract consists of a set of interacting smart contract modules:

- Registration Module
- Commitment Module
- Order & Delivery Module
- Test Module

The methods of the smart contract have been grouped into four modules. These modules are referred to as

- Account Management Module
- Data Catalogue Module
- Order & Delivery Module
- Contract Management Module

An overview of the functionalities is provided below.

The contract must store several categories of on-chain information, including senders, commitments, and orders. To do so in a structured and gas-efficient manner, it uses four dynamic arrays and two mappings. The arrays contain records generated and numbered by the main functions. In Solidity, such grouped records are called structs: each struct combines the related fields of one sender, commitment, reset request, or order. The mappings connect identifiers and addresses to the corresponding records, allowing the contract to locate the relevant information efficiently.

The BCP smart contract uses the following events:

- newSender
- newCommitment
- newOrder

- dataDelivered
- newRequest

Structs:

- Sender
- ResetPUK
- Commitment
- Order

Constructor

In Ethereum, a constructor is executed when a smart-contract instance is deployed. In Blockchain Presence, the constructor initializes four parameters.

- **Owner** is the address that deployed the contract. Several functions are restricted to this address (see Contract Management).
- **BCPGross** is a variable that accumulates realized fees and is initially set to zero (see Fee Management).
- **BCPActive** is a Boolean that controls whether the contract accepts new commitments.
- **BCPSafeLow** sets the minimum gas price accepted for a sender and is initially set to zero (see Data Catalogue).

2.2 Registration Module

2.2.1 Overview

The registration module is a key element of the on-chain authentication protocol. It allows information providers to register by specifying a pair of operational addresses, referred to as PIN and PUK, and (optionally) three recovery addresses (PUK2a/PUK2b/PUK2c). The registration data of the senders is stored in a dynamic array.

The registration module is responsible for the sender registration and address ownership. Each information provider that wishes to become a sender on the Blockchain Presence platform needs to be registered with at least two externally owned Ethereum addresses:

- PIN
- PUK

Among other functions, the BCP smart contract performs on-chain authentication of ordered data points. The sender's PIN address submits data to the Relay function within the BCP contract (see the Order & Delivery module). Beforehand, the PIN receives the gas costs paid by the receiver. The PUK is the sender's principal address: it receives the sender fee, where applicable, after a successful transmission. The sender can also use the PUK to replace the PIN through the ChangePIN function.

Finally, the PUK2 triplet consists of three addresses (PUK2a, PUK2b, and PUK2c). The sender is expected to control PUK2a, while the other two addresses may be held by trustworthy independent parties. Any two of the three addresses can authorize a reset of the PIN, PUK, and PUK2 triplet through ResetPINPUK. Suitable independent parties might include a board member, custodian, law firm, or other trusted fiduciary selected before registration.

The registration module consists of three main functions:

- NewSender
- ChangePIN
- ResetPINPUK

2.2.2 NewSender

To become a BCP sender, an information provider must register on-chain with the Blockchain Presence smart contract. This is the first step in establishing the sender's identity for other participants. The prospective sender must control at least three different addresses, which become the PIN, PUK, and PUK2a.

Two additional addresses, PUK2b and PUK2c, may be held by up to two trusted parties and must be selected by the sender before registration. In NewSender, the prospective sender records the initial three addresses together with these two additional addresses. The function then performs three checks. First, PIN, PUK, PUK2a, PUK2b, and PUK2c must be five distinct addresses. Second, the PUK must not already exist in the BCP on-chain database. Third, all supplied addresses except the PUK must differ from the account calling NewSender. If all three conditions are met, the sender is registered, the request counter is initialized to zero by default, and a unique senderID is created. This identifier remains unchanged even if the sender later changes the PIN or resets all addresses.

2.2.3 ChangePIN

Because the PIN is used by the Sender Convenience Applet (SCA), a sender may need to replace it for organizational or security reasons. ChangePIN provides this on-chain service. The sender calls the function from the corresponding PUK address and must also update the PIN used by the SCA. The new PIN must differ from the sender's other addresses and from every PIN or PUK already registered in the contract. It may, however, be an address in another sender's PUK2 triplet, allowing one sender to act as a trusted party for another.

2.2.4 ResetPINPUK

Although address relationships are secured on-chain, private keys may be stolen or compromised off-chain. ResetPINPUK therefore allows all addresses associated with a senderID to be replaced. The function requires authorization from at least two of the sender's three PUK2 addresses, so the sender must obtain confirmation from at least one trusted party. Both authorizing PUK2 addresses must submit the same senderID and replacement addresses. Their calls are stored and compared with the registered PUK2 triplet; once two valid approvals match, all addresses are replaced. Compromise of a single key is therefore insufficient to take over the sender's on-chain identity. Two colluding trusted parties could nevertheless do so, which makes their independent selection important. If a sender cannot identify suitable parties, Blockchain Presence could be designated as one of them, but this would shift part of the authentication authority to the platform operator.

2.3 Commitment Module

When an information provider has registered successfully as a sender he may want to offer his data points to potential receivers in the public field. Since receivers may want to order

their data points for dates lying in the future they have to rely on the senders confirmation to provide the ordered data on the corresponding date. To ensure that a sender is able to provide the ordered data point on the requested date we have to guarantee that the receiver cannot order his data point beyond a possible time frame set by the sender. Thus, each sender has to commit himself to send his provided data points up to a given time horizon. This is done by the NewCommitment function.

2.3.1 NewCommitment

A registered sender commits to deliver the data points listed in a CSV file until a specified future date. The commitment includes a description hash that identifies the file and its data points. A sender may maintain several CSV files in parallel. NewCommitment takes five arguments.

These are:

- the senderID to identify the sender that's committing himself
- the senderFee that each sender can set for his data points
- the gasPrice for sending the data point to the Relay function and further to the receiver
- the description hash that identifies the contained data points of a .csv file that gets provided by a sender and finally
- the date that consists of unix time until when a sender commits himself to deliver the data points contained in his .csv file.

Before a commitment is stored, the contract performs several checks. First, BCPActive must be true, indicating that the contract still accepts new commitments. The owner can set this Boolean to false through PhaseOut (see Contract Management), after which no new commitments are accepted. Second, the function verifies that the caller is the sender's PUK. Third, the commitment date must lie in the future. Finally, the proposed gas price must be at least the configured safe-low value. The owner may adjust this value as market gas prices change. If the requirements are met, the commitment is stored and receives a unique commitmentID. Similar to the senderID, this identifier remains unchanged if the commitment horizon is extended. A registry mapping connects the senderID to the commitmentID, and the newCommitment event informs the website of the new record.

2.3.2 HorizonExtension

When a sender now previously set his commitment horizon to be shorter than possible (let's say as a precaution) then he may want to extend the time horizon for the same .csv file as before. In this case the sender only would have to call the HorizonExtension function with his:

- senderID
- commitmentID
- new commitment horizon (unix time)

The function takes the senderID, commitmentID, and new commitment horizon as arguments. It first verifies that the caller is the sender's PUK registered for the supplied senderID. It then checks that the

new date lies later than the existing horizon. If both conditions are met, the commitment horizon is extended; otherwise, it remains unchanged. A sender may extend a commitment but may not shorten it, which protects receivers who rely on existing orders. The function then emits newCommitment so that the website can reflect the update.

2.3.3 SetSafeLow / GetCommitmentInformation

The Data Catalogue module also includes SetSafeLow and GetCommitmentInformation. SetSafeLow allows only the owner to set a minimum gas price intended to reduce the risk that the order functions or the low-level call in Relay fail because the sender supplied an insufficient gas price. A transaction using the minimum may take longer to be mined, and the appropriate level must be reviewed as network conditions change. Because the constructor initializes the value to zero, the function may remain unused without blocking other functionality. GetCommitmentInformation is a view function. When called off-chain, it does not consume gas. Given a senderID and commitmentID, it returns the current commitment horizon and description hash, allowing receivers to determine how long the sender's data may be ordered.

2.4 Order & Delivery Module

This module consists of two types of functions:

- Order functions
- Relay function

2.4.1 Order functions

The order functions create and store requests for data points. Before an order is accepted, several requirements must be met. The requested delivery date must fall within the relevant commitment horizon, which the contract obtains from the commitmentID. The receiver must also provide sufficient payment for the full delivery process, including the order call, SCA execution, Relay, the receiver's mailbox function, and any sender fee. If the payment is sufficient, the estimated gas costs are transferred to the sender's PIN. The contract then stores the receiver address and commitmentID and emits two events: newRequest notifies the website, while newOrder supplies the SCA with the information needed for delivery.

There are two principal families of order functions: OrderNow and NewOrder. OrderNow schedules immediate execution, whereas NewOrder allows the receiver to specify a future delivery time. Additional variants predefine selected parameters for convenience. For example, OrderNow_1_1_200K sets the row and column to one and maxGas to 200,000; NewOrder_1_1 fixes the row and column; and NewOrder_1hour schedules delivery one hour later. The variants reduce the number of parameters that a receiver must supply for common use cases.

2.4.2 Relay function

Relay receives data points submitted by a sender's SCA and forwards them to the relevant receiver. The function has four stages:

1. Authentication

2. Compensation
3. Delivery
4. Confirmation

Thus, the SCA has to provide the following arguments:

- orderID
- data
- statusFlag

Authentication

Relay first verifies that the SCA submitted the data from the sender's registered PIN address. Starting from the supplied orderID, the contract follows a three-step lookup:

- orderID => commitmentID
- commitmentID => senderID
- senderID => PIN

The process consults the three BCP modules to obtain the PIN address. Because the order record is used again later in Relay, it is read from storage; the sender and commitment records are copied to memory, which is more gas-efficient for temporary access.

Compensation & Delivery

The compensation stage checks the Boolean statusFlag to determine whether the requested data point was available.

- statusFlag = false → no data point received
- statusFlag = true → data point received

The subsequent procedure depends on this flag. If no data point was available, the payment remaining for the order is returned to the receiver. The contract then calls the mailbox function of the receiver's smart contract with the arguments received from the SCA, thereby notifying the receiver that delivery failed. If the data point was delivered as expected, Relay forwards the arguments to the receiver's mailbox function. After a successful low-level call, half of any sender fee is transferred to the sender's PUK; the other half remains in the BCP contract and is added to BCPGross (see Fee Management).

Confirmation

The final stage begins once delivery has completed, as indicated by the local Boolean sent. If sent is true, Relay emits dataDelivered to inform the website whether the order succeeded or failed. The corresponding orderID is then deleted, which recovers part of the gas originally spent to store the order.

2.5 BCP Test Module

This is the last module of the contract and contains two main functions as well as two view functions. The main functions are:

- PhaseOut
- Collect

PhaseOut can be called only by the contract owner and controls whether NewCommitment remains available (see Data Catalogue). It sets BCPActive to false, after which no new commitments can be registered. The constructor initializes BCPActive to true so that commitments are accepted until PhaseOut is called. Collect, like PhaseOut, is restricted to the owner. It transfers realized fees to the owner's address (see Fee Management).

The module also provides two view functions:

- GetBalance
- GetBCPGross

These functions expose different stages of the contract's income. GetBalance returns the total ETH held by the contract, including payments associated with unsettled orders. GetBCPGross returns realized fees that may be withdrawn without jeopardizing outstanding obligations. The owner can therefore call Collect at any time to transfer BCPGross rather than the entire contract balance.

2.5.1 Fee Management

When placing an order, a receiver specifies the requested data and pays the amount needed for the delivery process (Hermalin and Katz, 2004). This is an advance payment: at the time it reaches the contract, successful future delivery cannot yet be guaranteed. If the sender cannot provide the requested data, the SCA sets _statusFlag to false (see the Order & Delivery module). The receiver then receives the remaining payment back, less gas already consumed. For this reason, the owner must not collect fees before they are realized. BCPGross records fees only after successful delivery, and Collect transfers only that amount to the owner, avoiding liquidity problems for unsettled orders.

3. Sender convenience applet (SCA)

3.1 Overview

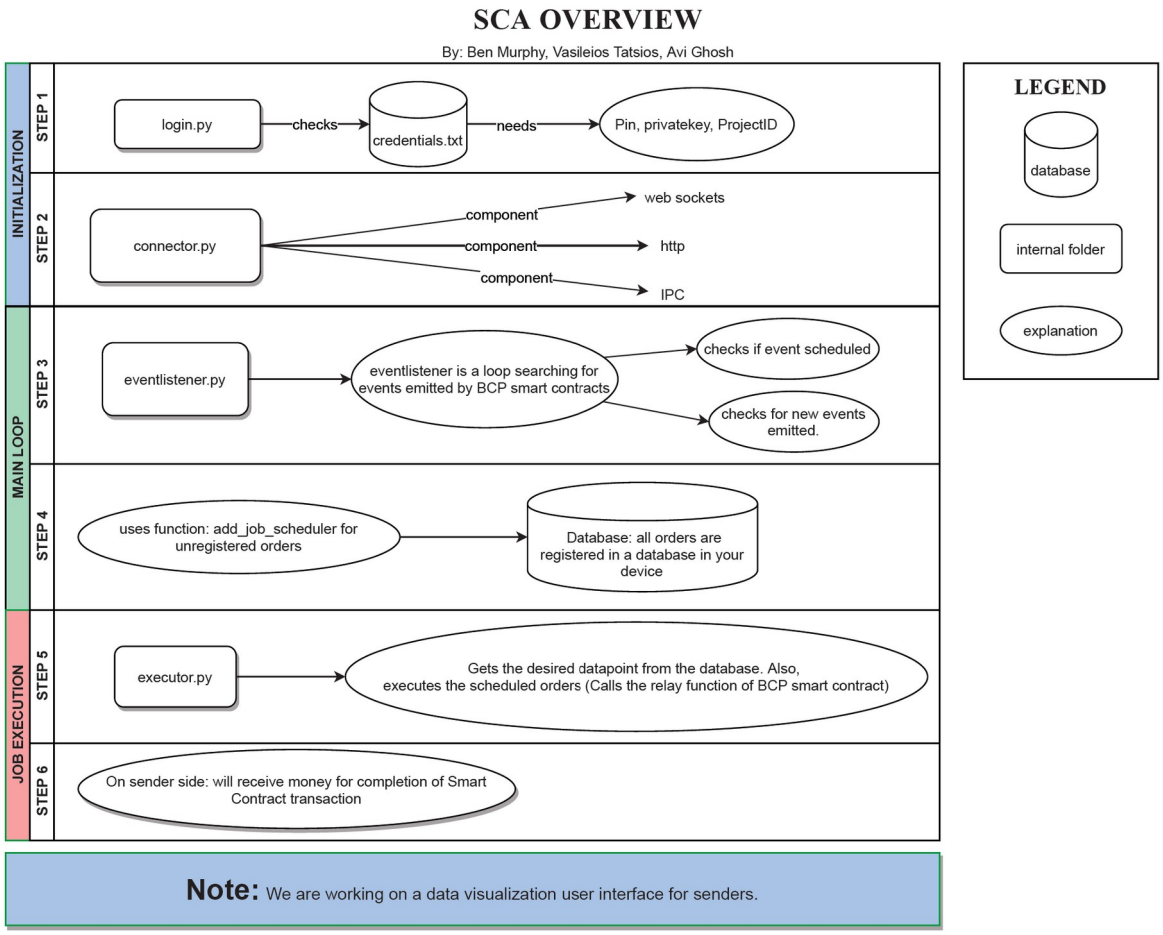
The SCA is a script that runs on the sender's server. In its implemented form, it can also run on an ordinary computer with a stable internet connection.

The SCA listens for events emitted by the Blockchain Presence smart contract and filters for orders relevant to the sender, identified by the registered PIN. It obtains the requested value from the

configured data source - for example, a CSV file or an application-specific reader - and checks it against predefined constraints. If the checks succeed, the status flag is set to true; otherwise, it is set to false. The resulting relay call is scheduled either immediately or for the future delivery time specified in the order.

When execution is due, the scheduler calls Relay with the relevant arguments and the gas parameters supplied by the order or current SCA configuration.

If the SCA stops unexpectedly, scheduled jobs are not lost. The scheduler maintains a jobs.sqlite file in which new jobs are added and completed jobs are removed. When the SCA restarts, it reads this file and resumes outstanding work.

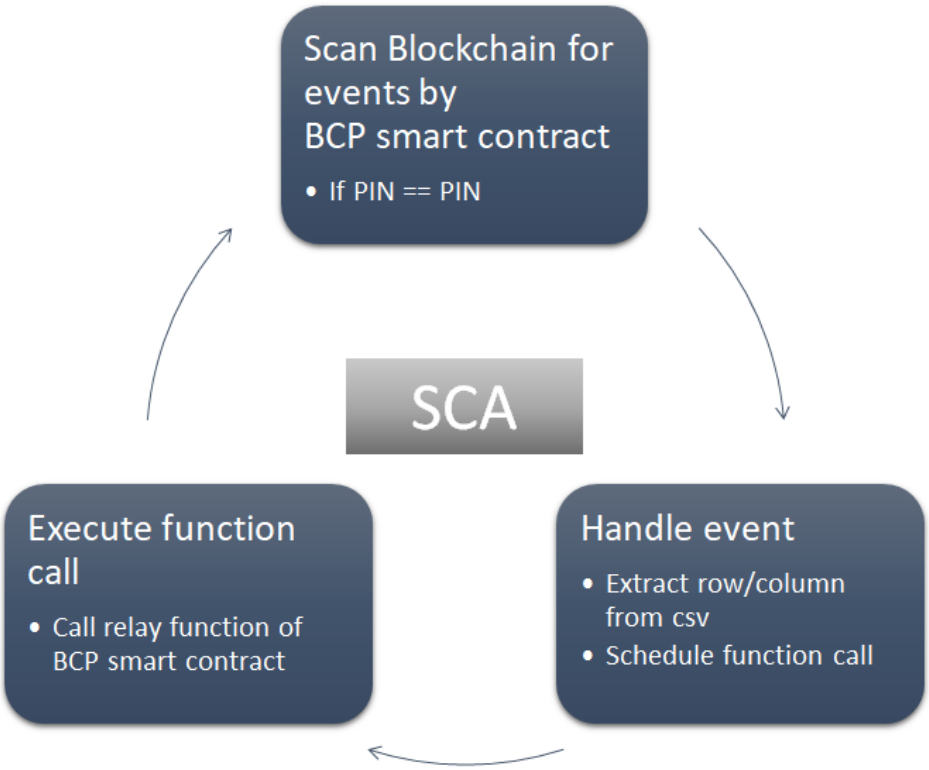


The CSV file is an extracted, reduced version of the sender's operational database. The commitmentID determines which file the default reader accesses. For example, if a sender's first relevant commitment has ID 000005, the corresponding file in the working directory is named BCP_000005.csv. An optional data-reader component can instead connect the SCA to another local or remote data source.

The sender does not need to know where his information is being relayed to. The data points are defined through the measures of:

- row

- column



The SCA has three main components:

- an event listener
- a scheduler
- a file reader

The event listener scans for events emitted by the BCP smart contract and filters out orders that are not relevant to the sender. Relevant events are passed to the scheduler, which plans execution for the time specified by the receiver. When that time arrives, the data reader obtains the requested value from the configured source and the SCA submits it to the BCP smart contract, which relays it to the receiver.

3.2 Initial prototype

The initial prototype transmitted orders through the input-data field of ordinary transactions. Two classes were developed to send and read those transactions, followed by an order-book class that scheduled future delivery. During development, the process was redesigned so that orders arrived through events instead. The order book was then replaced by a scheduler library, leading to sca_1.1.9.py.

3.2.1 sca_1.1.9.py

The discovery that orders would arrive as events rather than transactions led to an extensive redesign. The CSV-reader function was the main component retained from the initial prototype.

3.2.2 Connecting to a light node

To interact with the blockchain, the SCA must connect to a node. Several node providers are available; the prototype used Infura. During testing, it connected to the Ropsten network rather than Ethereum Mainnet. The connection details and network can be changed in the SCA configuration.

When using an Infura WebSocket endpoint, the URL must contain the correct WebSocket path. The sender's PIN and the Blockchain Presence contract address must also be included in the SCA's contract-address allowlist. The allowlist can be disabled, but doing so is not recommended.

3.2.3 Signing transactions on the backend

For autonomous operation, the SCA signs transactions programmatically rather than through a browser wallet such as MetaMask. The web3.py library builds and signs each transaction using parameters such as the nonce, value, gas limit, gas price, and the sender's private key, and then submits the signed transaction to the network.

3.2.4 Event reader

The event reader scans the blockchain for incoming events. A filter selects events whose PIN matches the sender and begins at the last recorded block. The query is repeated at short intervals. Asynchronous execution allows the event listener and event handler to run concurrently.

3.2.5 Scheduler

The scheduler plans jobs for immediate or future execution. Each job includes the orderID, data point, status flag, gas limit, and gas price. A persistent SQLite job store records scheduled work. When the execution time arrives, the scheduler removes the job from the store and calls Relay. If the SCA stops, the stored jobs remain available after restart.

3.2.6 Checkpoint systems

Two checkpoint mechanisms support reliable delivery. First, scheduled jobs are stored persistently in SQLite. Second, the event listener records the last processed block and the transaction hashes of handled events. After a restart, it scans from the saved block to the latest block and skips events whose transaction hashes are already recorded. In the version described here, a scheduled job may execute within a configurable grace period after its intended time; jobs missed beyond that period are cancelled.

3.2.7 Inter-functionality: full process

When the SCA is running, the end-to-end process is as follows:

When a receiver places an order, the BCP smart contract transfers the estimated gas funds to the sender's PIN and emits an event. The SCA continuously scans for events whose _PIN argument matches the sender's PIN. It extracts the event parameters and uses the row and column values to locate the requested data point in the configured data source. If the field is missing or does not contain an integer, the SCA returns the placeholder value 404 and sets the status flag to false. If validation succeeds, it sets the status flag to true. A successful delivery makes the sender eligible for the agreed fee, which is paid to the PUK; an unavailable data point instead results in reimbursement of the receiver, subject to gas already consumed. The SCA then interprets orderType to determine the execution parameters.

For order types 0, 2, 3, and 4, the gas limit and execution time are taken from _maxGas and _orderDate.

For order type 1 (orderNow_1_1_200K), maxGas is fixed at 200,000 and the execution time is taken from _orderDate.

After handling the event, the SCA writes the block number to prev_block.txt and records the transaction hash in sqlite_tx.db, as described under checkpoint systems. It then schedules the relay call with the orderID, data point, status flag, gas parameters, and execution time extracted from the event.

Over time, the stability of the SCA has been improved and further functionalities have been added.

3.3 SCA stability

To process incoming events reliably, the SCA should run continuously on the sender's server.

Connection recovery. The event listener catches WebSocket connection errors and enters a reconnection loop instead of stopping. Once the connection is restored, it resumes scanning from the last checkpoint.

Filter recovery and automatic restart. Node providers may delete event filters after a period of inactivity. In that case, the SCA creates a new filter and restarts the event listener. Most other node-side errors are handled in the same way, with two exceptions:

Timeout. If the SCA cannot reconnect within 24 hours, it reports a timeout and shuts down. This limit corresponds to the version's job-execution grace period.

KeyboardInterrupt. The keyboard shortcut Ctrl+C provides a controlled way to stop the SCA.

3.4 SCA functionality

In order to increase the sender convenience, the following functionalities have been added:

Initial interaction with the application:

Initial interaction with the application. The SCA includes a welcome section and allows previously stored credentials to be changed.

Private-key storage. For security, the SCA stores only an encrypted version of the sender's private key. The sender chooses the encryption password.

Possibility for API interaction

The SCA was initially designed for Excel files used as a sender database. To support other data sources without changing the main SCA logic, a separate data-reader component (data_reader.py) can be adapted by the sender. When activated, this component replaces the default file reader. The reference implementation includes examples for a local file system and the public Coinbase API.

The communication method in the earlier SCA version was predefined: WebSockets through Infura. To avoid dependence on a single provider and allow other configurations, web3_Provider.py lets the sender choose both the blockchain node and the web3 transport for each network. The original Infura WebSocket configuration remains the default.

Blockchain connection through different web3 providers and nodes

Nonce management. A transaction nonce identifies the sequence of transactions sent from an Ethereum address and prevents replay. Gaps can delay all subsequent transactions from that address, so careful nonce management is important for reliable SCA operation.

Improving management of the nonce transaction's parameter

The nonce is a property of the transaction-originating address and represents the number of transactions sent from that address (Patel, 2019). It makes each transaction unique, prevents replay attacks, and determines execution order. A missing nonce can block later transactions until the gap is filled.

In the version described here, the SCA waits for a transaction receipt before sending the next transaction. This prevents nonce conflicts but can be slow during periods of high activity. A more efficient nonce-management process was developed to execute jobs more quickly, but it was not included in the public SCA because it had not yet been tested sufficiently.

Prevent double job execution

Double job execution can occur if the same orderID is scheduled twice. To reduce this risk, the SCA stores the hash of each scheduled event in SQLite. At startup, it checks whether scanned events have already been handled and skips them if so. This works reliably for one SCA instance using one local database, but does not by itself prevent duplicate execution across separate instances.

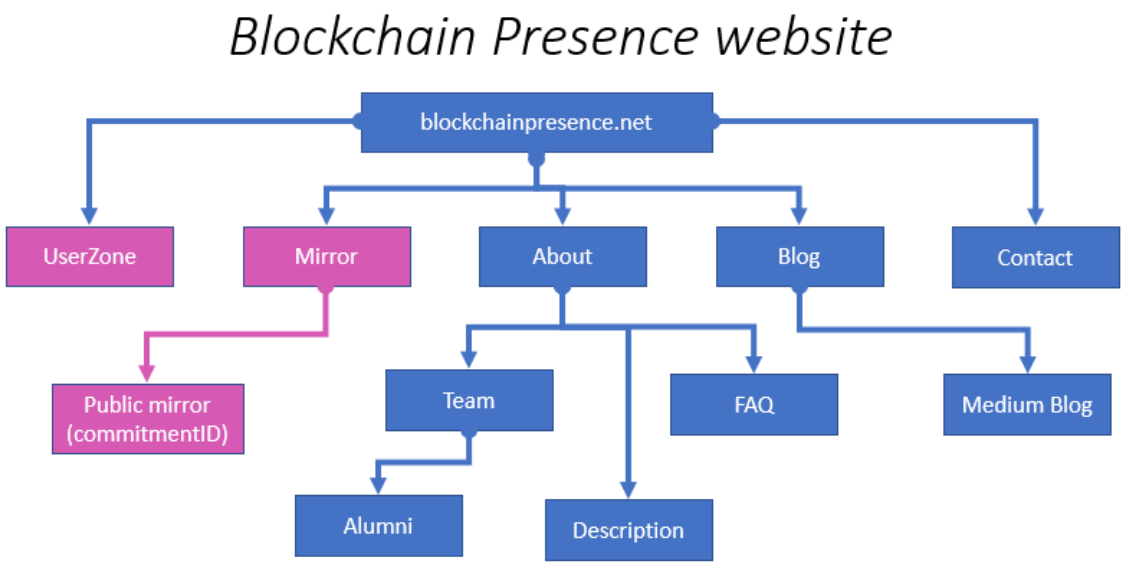
Checking already delivered orders. Because a local SQLite database cannot show whether another SCA instance has already delivered an order, the SCA also checks the blockchain and discards orderIDs that have already been completed.

Continuous checks for executed jobs. If two SCA instances use the same credentials, events returned by get_new_entries() could otherwise be scheduled by both. The SCA therefore checks newly received event hashes continuously, rather than only during startup.

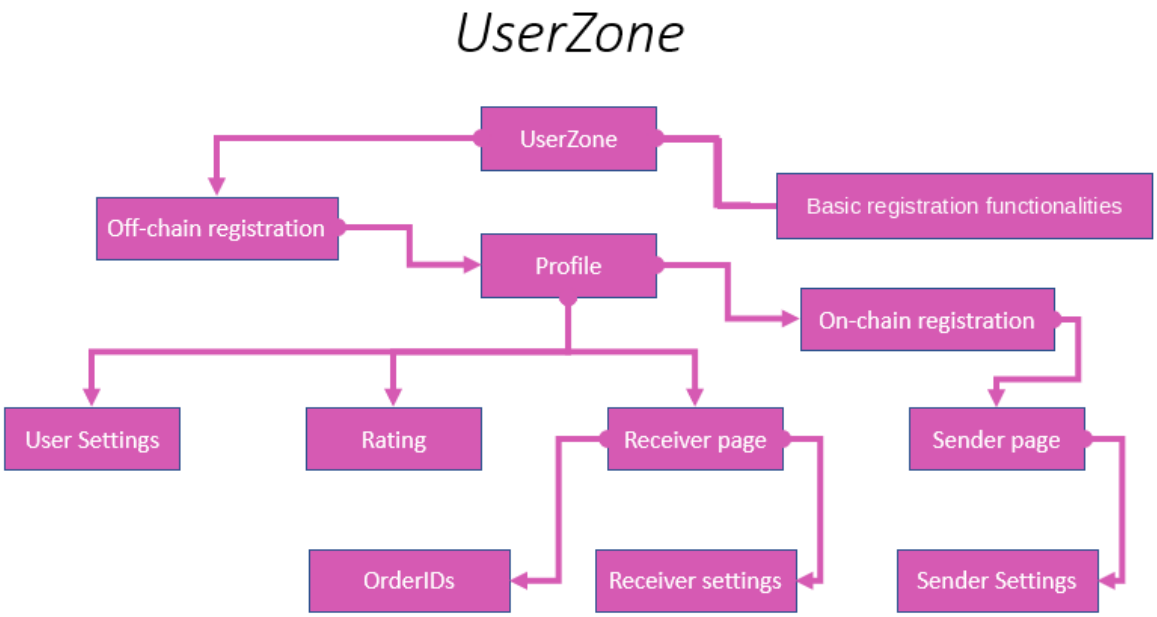
SCA activity test. If multiple SCAs run with identical credentials but separate local databases, each may execute the same incoming order. A possible long-term solution is a shared non-local database accessible to all instances. In the prototype, the sender can instead run a test order on Ropsten: if another SCA already delivers the same orderID, the testing instance detects the duplicate dataDelivered event and shuts down.

4. Website

An integral component of the *Blockchain Presence* platform is the website. The composition of the UserZone was left out on purpose. Its structure will be displayed on its own chapter. (Update!)



4.1 UserZone



4.1.1 Off-chain registration

Off-chain registration follows a conventional account-creation process. The user supplies a username, email address, and password, and verifies the account through an email link. The website also offers password reset and user settings. After verification, the user is redirected to a profile page.

- On-chain registration as a sender
- Registering a receiver smart-contract address and, where necessary, the deployment nonce used to derive the first contract address. This information allows the website to filter and display the user's orders.
- Providing feedback on senders in the form of a 5-star rating

4.1.2 On-chain registration and sender page

This page allows the user to register the authentication information for a senderID: the PIN, PUK, and optionally a PUK2 triplet. Registration is on-chain and therefore requires a wallet such as MetaMask and a small amount of ETH for gas. After registration, the user is redirected to the sender page, where the authentication information can be updated. The sender may also provide a short public description that is displayed in the mirror overview, allowing users to understand the sender without visiting the sender's complete website.

4.1.3 Rating

On this page the user can rate the senders. He has access to a dropdown, where all possible senders will be displayed. There he can choose a sender and rate it with a value from 1 to 5.

4.1.4 Receiver page

On the receiver page the user gets an overview of all his orderIDs. To be able to see them he needs to register his receiver smart contract address, which he will use to interact with the BCP smart contract. He can set this up on the receiver settings page.

4.2 The mirror pages

The overview and public mirror pages are accessible to everyone.

4.2.1 Public mirror

The **public mirror** provides an overview over senders and commitments. As the name suggests, the public mirror is accessible for any user, i.e., prior registration or login is not required for viewing the public mirror. The public mirror consists of two pages, the sender list and the commitment list. These pages are described in more detail below.

Sender list

On the overview page the user can see a table with:

- name of the company
- SenderID
- Link to his webpage
- 5 Star rating
- total number of “receiver-pulls”

  The public mirror provides an overview of senders and commitments. As the name suggests, it is available without registration or login.

Commitment list

This page lists all commitments, for example in a table with the following columns:

- **CommitmentID.** A running index, starting at 1. More recent commitments are shown at the top of the table.
- **SenderID.** (linking to the sender list)
- Time horizon. The final date (year, month, day) up to which data may be ordered. A sender can extend this horizon through the corresponding BCP smart-contract function.
- **Description.** A text area with hash code. The hash code is stored on-chain.

If possible, the description should be shown on a separate page.

Private mirror

A sender may filter all the commitments that concern his SenderID. The overview page shows a list of the orders.

- **OrderID.** A running index, starting at 1. More recent commitments are shown at the top of the table.
- **Receiver smart contract.**
- **Status information.** This should allow everybody to see where the order stands:
    - Ordered, but not yet delivered
    - Ordered and delivered
    - Ordered and delivered with negative flag

4.3 Download area

The GitHub repository provides material for senders and receivers.

4.3.1 Sender download area

This page allows users to download SCA.zip, which contains files such as:

- A readme file: readme.asc
- The Python source code of the Sender Convenience Applet: SCA.py
- A default data file that allows the BCP team to run tests: BCP_0000.csv
- BCP_0000.txt. A default description file

The page may offer some additional guidelines (or video) regarding how to install the sender convenience applet ("SCA") on the sender's server.

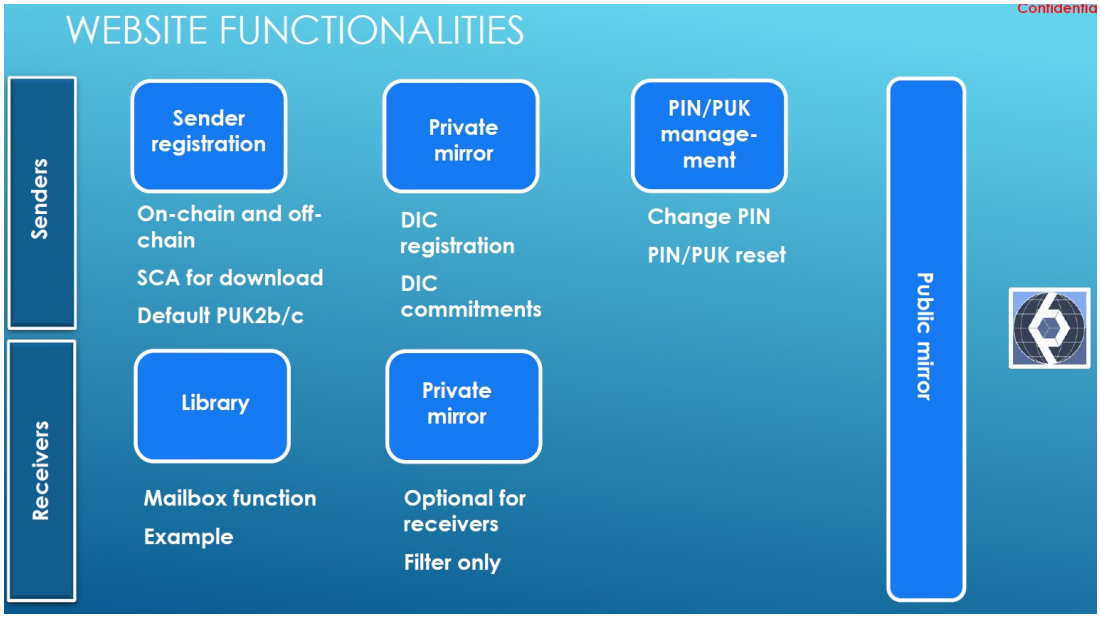
4.3.2 Receiver download area

This page provides examples of receiver smart contracts, such as TermDeposit.sol, together with explanations of the order and mailbox functions.

5. User perspective

This chapter describes the way in which users are envisaged to interact with the *Blockchain Presence* platform.

The following diagram summarizes the main website functionality:



5.1 Sender perspective

In this section, we describe the use of the *Blockchain Presence* platform by providers of information. In the sequel, these will be referred to as **senders**. The sender's interaction with the *Blockchain Presence* platform consists of three processes:

- User registration as a sender
- Description of data points through a DIC claim
- Delivery of data points via push

Below, we describe these processes.

5.1.1 Getting started

Registration

Sender registration takes place via the *Blockchain Presence* website. The registration consists of two steps:

OFF-CHAIN REGISTRATION

This is a standard process for setting up an online user account. CAPTCHA can be used to reduce automated abuse. The platform may perform a background check on a sender's identity and may suspend users who misuse its off-chain services. Such users could still access the on-chain contract directly, but without the convenience services provided by the website.

ON-CHAIN REGISTRATION

Before on-chain registration, the sender creates the required Ethereum accounts. Their balances may initially be zero, but the accounts used to submit registration transactions need a small amount of ETH for gas.

The sender then calls the registration function with the required addresses. The calling address and the function arguments become the PIN, PUK, and PUK2 addresses according to the contract's

registration rules. The contract derives a unique senderID and records the mapping from that senderID to the authentication addresses.

The addresses serve different purposes. The PIN is the operational address used by the SCA to submit data. The PUK can authorize a change of PIN. The PUK2 triplet can authorize a broader reset of the PIN and PUK, subject to the required multi-signature approval. The PUK2 triplet cannot be changed unilaterally.

The access information for these accounts, especially the private keys for the PUK and the PUK2 should be kept by the sender in a safe place.

Installation of the SCA

WINDOWS 10 ENTERPRISE

1. Add the MetaMask extension to Firefox or Chrome
2. Create at least two Ethereum accounts: PIN, PUK
3. Set up access to an Ethereum node provider
4. Create a folder for your data: <your path>\bcp_database
5. Install and run Docker with administrator privileges
    a. If a problem occurs, take care that Hyper-V is activated
    b. If that does not help, activate CPU virtualization in BIOS
6. Open PowerShell as administrator and run the following commands:
    a. docker login
    b. docker pull moerfi/blockchain_presence:sca
    c. docker container run –v <your path>:/bcp_database/ -it moerfi/blockchain_presence:sca
7. Provide PIN, private key of PIN, and Infura project number.

5.1.2 Entering data descriptions

A DIC claim is an offer to deliver a specific data point over a specified period.

A data identifier code (DIC) identifies the smallest unit of data that a sender offers to deliver. It is a prerequisite for representing the data on the Blockchain Presence platform. Registered senders may claim any number of DICs through the website.

To claim a DIC, senders log in to their private mirror. This page lists their claimed DICs and related information, while the public mirror shows a reduced public view.

After a DIC has been registered, the sender receives a confirmation email with a link to the public mirror page containing the description and ordering information. The sender may include this link, or a generated badge containing it, in other documents and webpages.

5.1.3 Making commitments

Registered senders may also publish data-source proposals on a public pinboard.

Automated delivery is performed by the Sender Convenience Applet (SCA), which the sender installs on a server or continuously connected computer. The SCA connects to an Ethereum node or remote client. Users of the automated process must rely on the SCA implementation to operate as documented.

5.1.4 Troubleshooting

Unavailable data

If a sender cannot provide an ordered data point for an unforeseen reason, the corresponding database entry can indicate that the data is unavailable.

Manual push

As an alternative to the SCA, a sender may deliver data manually through Remix.

A data point consists of a delivery address, DIC, value, and date. Upon receipt, the BCP smart contract checks whether:

(1) the DIC has been requested (trigger check), and if

(2) the DIC has been claimed by the sender (authenticity check).

If both checks are positive, the BCP smart contract forwards the data point to the receiver smart contract at the address supplied when the order was placed.

For secure delivery over the last mile, the receiver contract can implement a mailbox function that verifies the following conditions:

https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc#addressing-an-existing-instance

The reference library should provide a function that receiver contracts can incorporate. It should verify that:

- The address from which the data is delivered is that of the blockchain presence smart contract.
- The DIC is correct.
- The date is correct.

We will have to provide that function in a library.

The BCP process ends when the data is delivered to the receiver address. The receiver remains responsible for supplying the correct address and using the delivered information appropriately.

Compromised private keys

If the private key corresponding to the PIN is lost or compromised, the sender can change the PIN using the PUK. Because the PIN and its private key are used by the SCA, this operational key has a comparatively higher exposure risk.

If the private key corresponding to the PUK is lost or compromised, the PUK can be changed using the PUK2 recovery process.

If a PUK2 private key is lost or compromised, the broader recovery process requires approval from two of the three PUK2 addresses.

5.2 Receiver perspective

Receivers are users who obtain information through the Blockchain Presence platform. Smart contracts deployed by receivers are referred to as receiver smart contracts. A data point is identified by a senderID and a DIC.

There are two basic ways in which receiver smart contracts may interact with the *Blockchain Presence* smart contract:

- Pull function. A receiver smart contract may request a data point for immediate use.
- Order function. A receiver smart contract may request delivery of a data point at a specified future time.

Below, we describe the workings of the order function.

5.2.1 Order for delivery

A receiver can order a data point on-chain by calling an order function in the Blockchain Presence smart contract. The contract's Solidity interface specifies the required call.

Discussion. A receiver contract must check that the requested date lies within the sender's commitment period. Orders submitted after that period may not be fulfilled.

5.2.2 Use of the mailbox function (formerly callback function or library function)

The receiver smart contract must contain a mailbox function that the BCP contract calls during last-mile delivery. The website should provide a minimal reference implementation for download. Its purpose is to receive the ordered data point and verify that the call originates from the BCP contract.

The box below shows an example of a receiver smart contract.

```
pragma solidity >=0.4.0 <0.6.0;

contract ReceiverContract {

    mapping(uint256 => uint256) myOrders;

    address BlockchainPresence;

    function SetAddress(address BP) public {
        BlockchainPresence = BP;
    }

    modifier AddressCheck(){
        require(msg.sender == BlockchainPresence);
    _;
    }

    function checkResult(uint256 orderID) public view returns (uint256 value){
        return(myOrders[orderID]);
    }

    function callbackBP(uint256 orderID, uint256 _data) public AddressCheck {
        myOrders[orderID] = _data;
        mySenders[orderID] = msg.sender;
    }

    mapping(uint256 => address) mySenders;

    function checkAddress( uint256 orderID) public view returns (address Sender){
        return(mySenders[orderID]);
    }

}
```

5.2.3 Registration and private mirror

The user registration for the receiver is off-chain only.

The private mirror shows the orders placed by the receiver or by the receiver's smart contracts. Because orders are visible on the blockchain, the underlying information is not confidential. The mirror nevertheless provides a convenient private filter over the receiver's own contracts and orders.

There are a number of functionalities. Specifically, the receiver can:

(1) check the status of each order;

(2) view statistical information about the likelihood of timely delivery and, where appropriate, receive warnings;

(3) adjust the gas price used for delivery, subject to the resulting cost; and

(4) indicate demand for particular data sources through the pinboard.

After selecting the required data point, the receiver calls new_order and supplies the payment associated with the selected sender and service parameters.

Once payment is received, new_order creates an order and performs three tasks:

1) It checks whether the order is feasible, including whether the sender committed to deliver the data at the requested time;

2) It records the order in a dynamic array; and

3) It notifies the SCA of the order.

The order contains the information needed to identify, retrieve, and deliver the correct data point. Its arguments include:

- Sender
- DIC
- Delivery address
- Delivery time

Although the delivery address may sometimes be inferred from msg.sender, the caller of the receiver contract may differ from the intended destination. Supplying the delivery address explicitly therefore makes delivery more reliable.

After creation, the order is stored on-chain.

The SCA running on the sender's system receives the order and retrieves the requested data. At this stage, the sender is responsible for matching the correct data point to the order.

To increase confidence in the delivered information, Blockchain Presence envisages working with senders that meet appropriate reliability and collaboration standards.

Blockchain Presence cannot guarantee the sender's performance. The receiver must therefore assess the sender before placing an order.

At the requested time, the SCA supplements the order with the data point and submits it to Relay in the BCP smart contract.

Recording the delivery process on-chain provides an additional audit trail in case a problem arises before the data reaches the receiver contract.

Relay checks that the received data corresponds to the recorded order. The SCA calls Relay through web3 and includes the data point in the transaction payload.

The SCA sends:

- Order ID
- Data point

Relay first verifies the identity of the submitting account. The lookup has three steps:

1. Using the Order ID to find the Sender ID in the order array
2. Using the Sender ID to find the Sender PIN in the sender array
3. Comparing the message sender to Sender PIN

If authentication succeeds, Relay emits an event recording that the order and data point arrived at the BCP contract. It then calls the receiver's mailbox function and forwards:

- Order ID
- Data point

The mailbox is implemented in the receiver smart contract and accepts delivery from the BCP contract.

After the mailbox call succeeds, the BCP contract emits a further event confirming delivery for the orderID. The SCA receives the transaction result, and the on-chain authentication process provides evidence of a secure transmission from the registered sender to the receiver contract.

The receiver convenience tool consists of two elements:

- Order function
- Mailbox function

These elements will be described in more detail below.

5.3 Owner perspective

5.3.1 Deployment
The owner deploys the Blockchain Presence smart contract and thereby initializes the system. The constructor records the owner address and the contract's initial parameters.

5.3.2 Collection of revenues
Using the account that deployed the BCP contract, the owner may call Collect through Remix. Collect transfers fee income recorded as realized at the end of confirmed deliveries. Only income already earned through successful deliveries can be withdrawn.

5.3.3 Phase-out
To transition to a new version, the owner may call PhaseOut through Remix. This sets a flag that prevents new commitments while allowing the contract to continue processing orders covered by existing commitments.

6. Implementation issues

6.1 Testnet implementation
Given the sequential relationship between registration and commitments, on the one hand, and orders and delivery, on the other, the platform can be operated in three ways:

6.1.1 Testnet operation

A copy of the primary contract can be deployed on a test network, allowing senders and receivers to test the platform without using mainnet funds.

6.1.2 Order and delivery testing after mainnet registration and commitment

After testing, a sender may register and make commitments on mainnet. To support subsequent order-and-delivery testing, the mainnet and testnet deployments can operate in a primary-replica model. The backend filters registration, commitment, and horizon-extension events from the mainnet contract and reproduces them in the testnet contract through web3 calls.

6.1.3 Mainnet operation

6.2 Updating
Because the Order & Delivery Module calls view functions in the Registration & Commitment Module, the Order & Delivery Module can be updated without losing registration and commitment data.

7. Acknowledgements

The development of the Blockchain Presence DApp benefited greatly from the support of present and former members of the University of Zurich blockchain project.

8. Glossary

CSV (Comma Separated Value)

A file containing data values separated by a delimiter, such as a comma. It can serve as an intermediate format between a sender's database and the SCA.

Light node

“A light client or light node is a piece of software that connects to full nodes to interact with the blockchain. Unlike their full node counterparts, light nodes don’t need to run 24/7 or read and write a lot of information on the blockchain. In fact, light clients do not interact directly with the blockchain; they instead use full nodes as intermediaries. Light clients rely on full nodes for many operations, from requesting the latest headers to asking for the balance of an account.” (Sardan, 2018)

MetaMask

A crypto wallet (MetaMask, 2020) used to manage Ethereum accounts, sign transactions, and pay the gas costs of smart-contract calls.

Public Key/PIN/ Private Key

An externally owned account (EOA) is controlled by a private key and identified by a public address derived from it. In the Blockchain Presence design, a sender uses several accounts with distinct roles, including PIN, PUK, and PUK2 addresses. The SCA uses the designated PIN account for operational transactions.

Python

A general-purpose programming language widely used for server-side applications and data processing.

Remix

A browser-based integrated development environment used to write, compile, deploy, and interact with Solidity smart contracts.

Ropsten Network

A former Ethereum test network used to test smart-contract interactions without mainnet ETH. Ropsten was used by the prototype described in this paper.

SCA (Sender Convenience Applet)

One of the main components of the Blockchain Presence platform. It listens for orders emitted by the BCP smart contract, retrieves the requested data point, and submits it at the requested time.

Solidity

The principal programming language used to write smart contracts on Ethereum.

A. References

Antonopoulos, A.M., Wood, G. (2018), *Mastering Ethereum: Building Smart Contracts and Dapps*, O'Reilly Media.

Buterin, V. (2014), *Ethereum---A next-generation smart contract and decentralized application platform*, White Paper, 3(37).

Cong, L.W., He, Z. (2019), Blockchain disruption and smart contracts, *Review of Financial Studies* **32**, 1754-1797.

Hermalin, B., Katz, M. (2004), Sender or Receiver: Who Should Pay to Exchange an Electronic Message?, RAND Journal of Economics 35, pp. 423-448.

Ivanov, M. (2010), Communication via a strategic mediator, *Journal of Economic Theory* **145**(2), 869-884.

Mechtenberg, Lydia, and Johannes Münster. "A strategic mediator who is biased in the same direction as the expert can improve information transmission." Economics Letters 117.2 (2012): 490-492.

Raskin, M. (2017), The law and legality of smart contracts, Georgetown Law Technology Review 1, 305-341.

Szabo, N. (1997), Formalizing and securing relationships on public networks, *First Monday*.