Fr8 Protocol Technical Overview

Sloane Brakeville, Barry Glasco, Yev Spektor

14 November 2018

Contents

1 Abstract

Logistics as an industry has been in dire need of neutral, globally accepted protocol for connecting information systems. Current standards like EDI are beneficial, but as point-to-point solutions they fail to capitalize on network effects. The Fr8 Network is proposing Fr8 Protocol, a collection of rules, behaviors, and formats that specify a communication standard between two or more nodes on a network. Adoption of the protocol is driven by cost savings and efficiency gains powered through a tokenized economic engine.

2 Introduction

The complexity of global logistics cannot be understated. The result of decades of independent initiatives by organizations with similar needs is a mess of isolated technology systems that demand more and more scalability and responsiveness. Now, with the desire for Internet of Things adoption, track and trace, Co-modality, Green Corridors and Supply Chain Security, many groups have proposed interoperability solutions.

Electronic Data Interchange (EDI) was the first big initiative of many to bring digital innovation into logistics in an effort to reduce the amount of manual processing. Work on the standards began in the 1960s and continues to be widely used. However, weaknesses are abundant: maintenance and administrative overhead prevents adoption by Small and Medium Enterprise (SMEs), and huge complexity of messages with overloaded feature sets to name a few.

For example, the European Union has proposed a marriage of about a dozen autonomous research and development projects including FREIGHTWISE, e-Freight, INTEGRITY, Smart-CM, SMARTFREIGHT, EURIDICE, RISING, and DiSCwise

into the Common Framework. Progress is ongoing but output seems to have dwindled.

United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) is an intergovernmental body for the United Nations Economic Council for Europe (UNECE). The objective of this body is to publish works that facilitate efficient and standardized trade. Although promising and largely applauded, this standards stack shows enormous potential with its design and governance structures. Though potential is not enough, this stack is not widely disseminated[1].

GS1, a global not-for-profit standards body with an impressive track record of bringing in standards for global business, has contributed as well with their Logistics Interoperability Module (LIM). The aim of the Logistics Interoperability Model (LIM) is to establish business interoperability, the capability to run business processes seamlessly across organizational boundaries, in the Transport and Warehousing business processes[2]. Adoption data is limited so results of this standard remain to be seen.

What is clear and evidenced by the growing number of initiatives to establish standards for logistics is the enormous upside of operating on a globally recognized protocol. However, a massive hurdle has been in place preventing dissemination. As described in Integrity of Supply Chain Visibility: Linking Information to the Physical World:

> "Regulatory compliance in international trade can be enhanced by facilitating electronic exchange of trade documents to increase the supply chain visibility. Crucial for acceptance of the supply chain visibility concept is *trust in the reliability of the data*[3]".

For the first time since these multitude of projects began there is a solution to the need for trust in reliability of data: blockchains. While trust is only one problem facing global logistics, solutions in supply chain visibility will greatly reduce the nearly half a trillion dollars of counterfeit and pirated goods imported annually throughout the world[4].

Fr8 Protocol as described bridges the gap between the efforts of standards like EDI, RosettaNet, UN/CEFACT, and GS1's LIM; eliminating the deficiencies of each and incorporating the latest in blockchain advancements. As architected, stakeholders in the logistics industry will not need to cast aside ongoing work and can integrate with Fr8 Protocol modularly.

[1] https://www.alexandria.unisg.ch/30346/2/nceb06Final.pdf

[2] https://www.gs1.org/docs/tl/LIM_Foundation_Report.pdf

[3] https://link.springer.com/chapter/10.1007/978-3-642-31069-0_29

[4] http://www.oecd.org/industry/global-trade-in-fake-goods-worth-nearly-half-a-trillion-dollars-a-year.htm

3 Architecture

Below you can find Fr8 Protocol's high level architecture:

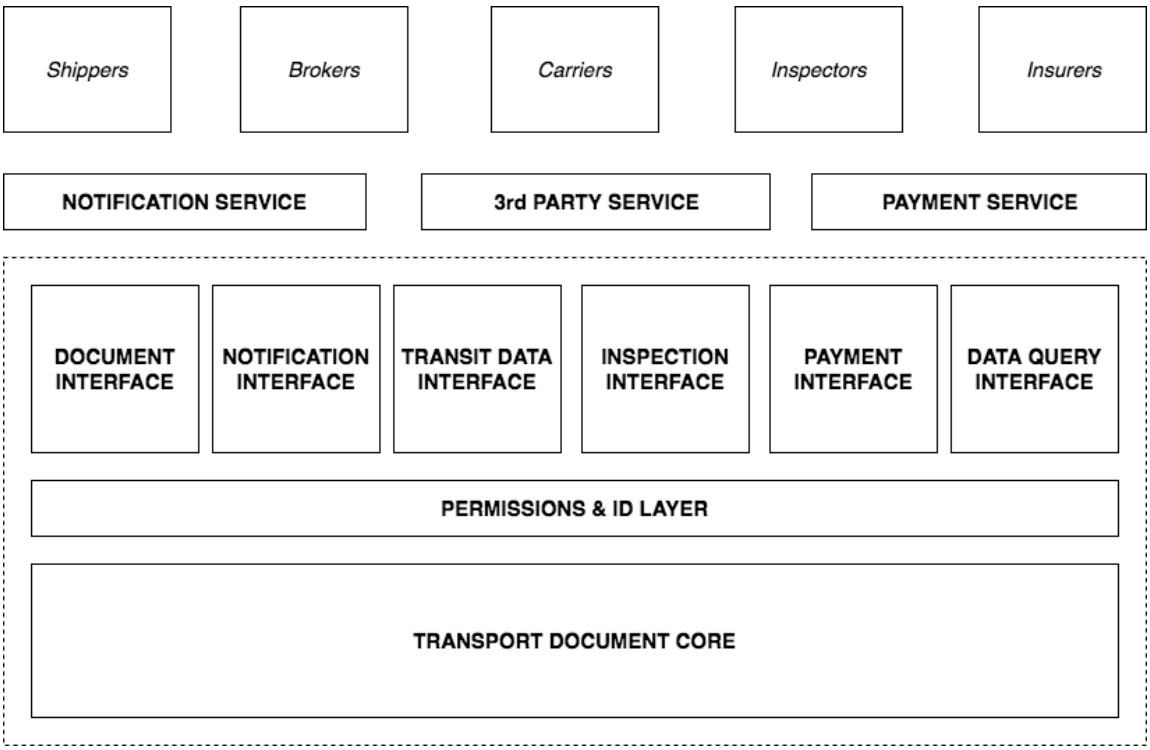

Figure 1: Fr8 Protocol Architecture

At the base, we have the Transport Document Core. This Transport Document Core stores and manages Fr8 Protocol's objects, their associations, and contains logic that connects interface functionality to the underlying object data.

Fr8 Protocol is organized by objects such as Shipments, Pallets, Documents, Inspections, Transit Data Collections, and more. Each of these objects contain data and associations. For example, Pallet could contain data describing its contents, ownership, tracking id's, and a reference to the Shipment to which it belongs. The Transport Document Core is responsible for both storing these objects and translating Interface requests into meaningful accessor and processing functions on these objects.

Incoming requests pass through the Permissions and ID Layer. Authentication is handled completely on chain using asymmetric cryptography. This layer handles the

auth "handshake", identifies the user, checks permissions, and validates or rejects the request.

Fr8 Network provides interfaces for user groups to interact with their shipments on the protocol. They are the Protocol's exposed endpoints. Each of these interfaces handles a specific function of the protocol and communicate via a REST API and Webhooks, depending on if the request is incoming or outgoing.

The REST API will be written in Node.JS, hosted redundantly and globally across multiple availability zones, and will use established scalable database services such as Aurora. Services connect with interfaces to provide application level functionality. While interfaces establish a standard for interacting with shipments on the protocol, the services enable companies to keep and use their existing processes with Fr8 Protocol.

Fr8 Protocol was design with modularity and flexibility in mind. By separating services from the protocol interfaces, Fr8 Network excels at interoperability and freedom of choice instead of imposing a new standard and forcing a business to change operations. This modular approach begins at the integration level, meaning that adopting Fr8 Protocol is not an "all-or-nothing" situation. Businesses are free to begin using Fr8 Protocol for just one of its capabilities, such as managing and storing GPS data or handling critical documents.

3.1 What is an Interface?

The interfaces of Fr8 Protocol provide methods of interacting with the Protocol and each conforms to an application programming interface (API) request format, data standard, and procedure to read and write data. A Representational State Transfer (REST) API is exposed for incoming messages and Webhook, or HTTP Callback, functionality is provided for subscription and outgoing message functionality. The APIs allow for modular component level development against the Protocol. Later in this document, we will discuss the purpose and functionality of each interface.

3.2 What is a Service?

Services plug into interfaces to provide application-level functionality. In essence, Interfaces provide lowest-level protocol interaction. The Services layer turn these low-level interactions into powerful logistics and business software. Services make raw protocol data accessible, useful, and meaningful to applications, plug-ins, and enterprise resource planning (ERP) systems.

We encourage 3rd party developers to create new and innovative services to sit

on top of Fr8 Protocol. The token model, combined with the services layer, allows for plugin and add-on business models to operate on the protocol. This creates market forces that incentivize Fr8 Service development. Amazon Web Services and Heroku have had great success with such a relationship with the 3rd party plugin developers. Likely candidates for early Services include data visualization tools, payment processing, and IoT platform bridges.

Upon launch, Fr8 Protocol will release its own Notification Service. This services listens, schedules, routes, and ensures the integrity of notification events from the Notification Interface. Encrypted notifications will be sent to the Notification Service and packaged with an array of required and optional recipients. For required recipients, the Notification Interface will await a response acknowledging successful delivery. Fr8 Protocol Notification Service will support SMS, email, Apple Push Notifications, Google Push Notifications, MQTT, RSS, and custom notification standards.

3.3 Blockchain Usage

Blockchain is used throughout the protocol to prove existence, authenticity, and integrity of all shipment data. While documents, metadata, transit data, and such are store off-chain, an immutable fingerprint of the data in the form of a SHA-256 hash is stored on-chain. The blockchain inherently keeps an immutable changelog and accesslog of the shipments. This methodology keeps sensitive business data private. Certain objects in the Transport Document Core have a hybrid existence and utilize both on- and off-chain storage. Identity, permissions, and authentication are handled on-chain.

3.4 Token Usage

Tokens are used and exchanged throughout Fr8 protocol. Fr8 Protocol and its interfaces require large-scale IT infrastructure. Uploading and storing data points generate several IT costs, such as hosting, databases, and maintenance. For this reason, Fr8 Protocol Tokens are required to upload data. However, it must be said that it would be impractical and prohibitively expensive for a company to attempt and implement this functionality itself, instead of subscribing to Fr8 Protocol.

A fee is charged to create a new shipment on Fr8 Protocol and for certain API requests. Fr8 Protocol fees operate in a way similar to many blockchain ecosystems including Ethereum. Fees occur every time data is written to the protocol, and reading data from the protocol is free. Services may also charge Fr8 Protocol tokens for

usage, but the fee structure is up to the 3rd party developers. By allowing developers to be directly compensated for usage of their Service on Fr8 Protocol, powerful community incentives are created that encourage expanding the functionality of Fr8 Protocol.

The Fr8 Protocol will not initiate or create the off-chain shipment object until the minimum number of Fr8 Tokens are sent to the Shipment Contract. Interfaces and Services will pull from this pool of tokens throughout its lifetime. Fr8 Protocol will provide a tool to estimate the number of tokens required for a shipment.

4 Fr8 Protocol Core Components

4.1 Transport Document Core

The Transport Document Core serves as the hub of Fr8 Protocol and is effectively centra controller, where all of a shipment's relevant data is stored and where all of the connectivity between parties happens. It knows everything about a particular shipment, who can view its contents, and who can change them. It uses the tools it has access to in order to achieve the goals of the shipment. This core is responsible for handling associations between objects on Fr8 protocol.

4.2 Core Objects

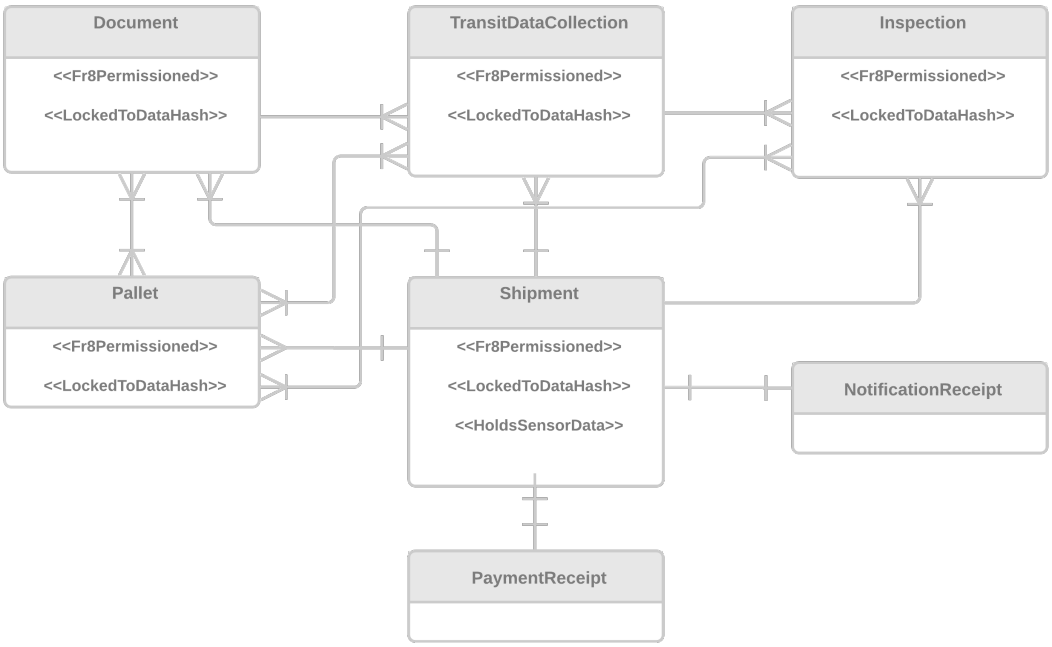

Figure 2: Fr8 Protocol Core Objects and Interfaces

4.2.1 Shipment

A shipment exists both on- and off-chain. Private and sensitive data is stored securely off-chain, while that data's hash is stored on-chain. On-chain, we store the shipment transit data hashes and permissions. Shipment details will be stored off-chain, but a SHA-256 hash data will be stored on-chain. Details include terms and conditions, description, tracking id's, required approvals, and other metadata. The Shipment Object also contains configurations for the Protocol Interfaces and any external services.

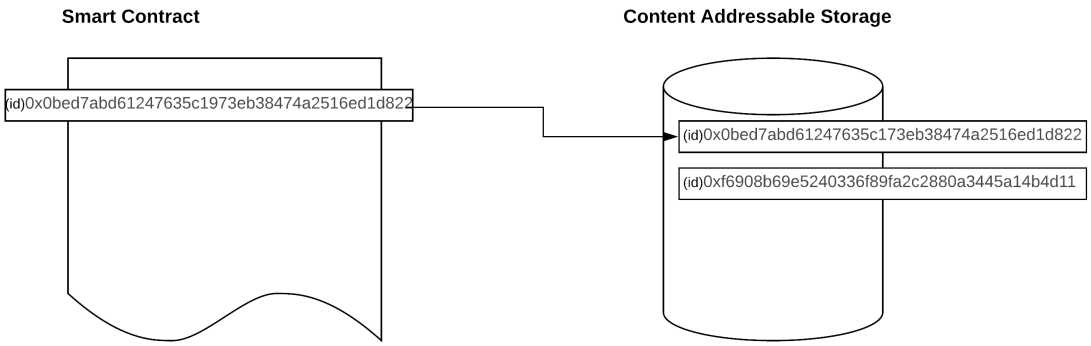

Figure 3: Fr8 Protocol Content-Addressable Storage

The off-chain storage is using what is known as Content-Addressable Storage Pattern (CAS)[5]. We will Store the data off-chain in a content-addressable storage system and store the reference in the smart contract. Clients using the smart contract can retrieve the hash of the data. Then, they can verify the data correctness by hashing the off-chain data to verify the hash correctness. This is an elegant solution to the privacy issues surrounding sensitive business data since no readable data is stored on the public blockchain. Here is how a client application would create a new shipment on the Fr8 Protocol:

1. Client deploys Shipment smart contract. The contract address will be returned.
2. Client sets permissions and data uploaders on Shipment smart contract.
3. Client sends Fr8 Tokens to Shipment smart contract.
4. Client POSTs to /api/document/shipment to create and register Shipment on Fr8 Protocol.
5. Client updates Shipment Smart Contract dataHash property with SHA-256 hash of Shipment data.

[5] `https://www.usenix.org/legacy/events/usenix03/tech/tolia.html`

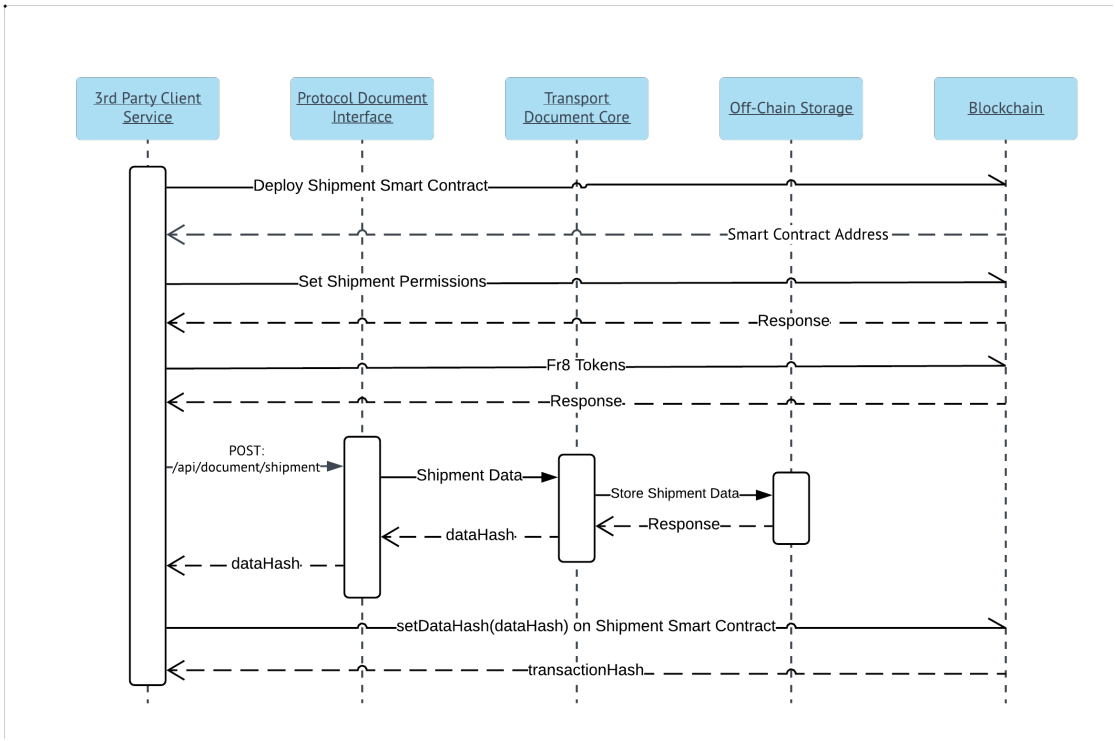

Figure 4: Fr8 Protocol Shipment Creation

```
/**
  * Fr8Shipment Class Interface
  */

contract Fr8Shipment is Fr8Permissioned, LockedToDataHash,
   HoldsSensorData {
  bool public hasBeenActivated;
  bool public hasBeenTerminated;
  bytes32 public PaymentReceiptsHash;
  bytes32 public NotificationReceiptsHash;

  modifier isActive();

  function addSensorDataUploaders(address[] _uploaders)
                                  public onlyEditor, isActive;
  function removeSensorDataUploaders(address[] _uploaders)
                                  public onlyEditor, isActive;
```

```
17
18    function activate() public onlyEditor;
19    function terminate() public onlyEditor, isActive;
20    function setPaymentReceiptsHash(bytes32 _datahash)
21                                    public onlyProtocol, isActive;
22    function setNotificationReceiptsHash(bytes32 _datahash)
23                                    public onlyProtocol, isActive;
24    function setDataHash(bytes32 _datahash) public onlyEditor,
          isActive;
25 }
```

Listing 1: Fr8Shipment.sol Interface

4.2.2 Pallet

A Pallet belongs to a Shipment, and a Shipment has multiple pallets. A Pallet can be transferred from one shipment to another. Like a Shipment, a Pallet follows a hybrid model and exists both on- and off-chain. On-chain, we store a pallets' transit data hashes and permissions. Pallet details and its associated shipment address will be stored off-chain, but the SHA-256 hash of the flattened, stringified data will be stored on-chain. Details include description, tracking id's, and other metadata.

```
1  /**
2    * Fr8Pallet Class Interface
3    */
4
5  contract Fr8Pallet is Fr8Permissioned, LockedToDataHash,
       HoldsSensorData {
6    function addSensorDataUploaders(address[] _uploaders) public
         onlyEditor;
7    function removeSensorDataUploaders(address[] _uploaders)
8                                       public onlyEditor;
9
10   function setDataHash(bytes32 _datahash) public onlyEditor;
11 }
```

Listing 2: Fr8Pallet.sol Interface

4.2.3 Document

A Document is a generic object that represents any type of document involved in shipping. On Fr8 Protocol, Documents support standard JSON text and media attachments such as images and pdf's. A single Document may have multiple attachments. Documents are stored on- and off-chain. Attachments and private data

are stored off-chain, while their hashes are store on-chain. Document permissions are handled on-chain as well. A Document may belong to a Shipment, Pallet, Inspection, or another Document.

```
/**
  * Fr8Document Class Interface
  */

contract Fr8Document is Fr8Permissioned, LockedToDataHash {
  struct AttachmentPair {
    string attachmentId;
    bytes32 attachmentHash;
  }

  AttachmentPair[] public attachments;

  function setDataHash(bytes32 _datahash) public onlyEditor;

  function addAttachments(string[] _attachmentIds,
                          bytes32[] _attachmentHashes)
                          public onlyEditor;
  function removeAttachments(string[] _attachmentIds) public
     onlyEditor;
}
```

Listing 3: Fr8Document.sol Interface

4.2.4 Inspection

Inspection objects are generic objects that represent forms requiring approval. Inspections inherit the Fr8Permissioned class and editors are able to Approve or Reject inspections on-chain. Inspections can have Document objects associated with them and belong to a Pallet or a Shipment. Inspection permissions are stored on-chain. Sensitive and private inspection data, as well as associations are stored off-chain.

```
/**
  * Fr8Inspection Class Interface
  */

contract Fr8Inspection is Fr8Permissioned, LockedToDataHash {
  mapping (address => uint) public approvals;

  function setDataHash(bytes32 _datahash) public onlyEditor;

  function approve() public onlyEditor;
```

```
11    function reject() public onlyEditor;
12  }
```

Listing 4: Fr8Inspection.sol Interface

4.2.5 TransitDataCollection

TransitDataCollection is an object containing a collection of many time-series sensor data points. A collection can be composed of one sensor or several sensors. This object also contains a unique id, description, and supports custom key-value pairs and properties. Data is uploaded to Fr8 Protocol through the Transit Data API. The upload process will be described in more detail in another section, but here, it is important to mention Fr8 Protocol will support data uploads from any sensors. Because Fr8 Protocol is designed in a modular way, a devices simply to needs to post to the REST API endpoint and send data in a standard JSON format. This data can be uploaded by a sensor, compound device, or platform with internet connectivity.

The collection's description and custom key-value pairs can be modified at any point. However, once a time-series data point collection has be uploaded, the data points cannot be altered or replaced in any way. A cryptographic hash of the time-series byte data is stored on-chain, in the smart contract of the object this data describes.

4.2.6 PaymentReceipt

A PaymentReceipt object is used to capture the result of a Payment transaction. Each object has a unique id created for it upon object creation. This object is not mutable and cannot be modified. Whenever a payment is processed by a service, the Payment Interface expects a result, or receipt, to be returned. The receipt contents will vary between different Payment Services, but in general contain a transaction id, whether or not the payment was successful, and payment metadata. This object supports custom key-value pairs. A cryptographic hash of the byte-level data of the entire object is stored on-chain, in the smart contract to which this payment belongs.

4.2.7 NotificationReceipt

A NotificationReceipt object is used to capture the result of a Notification event. Each object has a unique id created for it upon object creation. This object is not mutable and cannot be modified. Whenever a Notification Event is emitted to a service, the Notification Interface expects a result,or receipt, to be returned. The receipt contents will vary between different Notification Services, but in general

contain an notification id, status, and other metadata. This object supports custom key-value pairs. A cryptographic hash of the byte-level data of the entire object is stored on-chain, in the smart contract to which this notification belongs.

4.2.8 Common Interfaces

Several Core Objects such as Shipments and Pallets inherit one or more of three Fr8 Protocol Common Classes. The interfaces for these classes can be found below.

```
/**
  * Fr8Permissioned Class Interface
  */

contract Fr8Permissioned {
  address public protocolAddress;
  mapping (address => bool) public owners;
  mapping (address => bool) public editors;
  mapping (address => bool) public readers;

  modifier onlyProtocol();
  modifier onlyOwner();
  modifier onlyEditor();

  function addOwners(address[] _owners) public onlyOwner;
  function addEditors(address[] _editors) public onlyOwner;
  function addReaders(address[] _readers) public onlyOwner;
  function removeOwners(address[] _owners) public onlyOwner;
  function removeEditors(address[] _editors) public onlyOwner;
  function removeReaders(address[] _readers) public onlyOwner;
}
```

Listing 5: Fr8Permissioned.sol Interface

The Fr8Permissioned class is responsible for controlling all access and permissions for shipment data.

```
/**
  * LockedToDatahash Class Interface
  */

contract LockedToDataHash {
  bytes32 public dataHash;

  function setDataHash(bytes32 _dataHash) public;
}
```

Listing 6: LockedToDataHash.sol Interface

The LockedToDataHash class lets objects on Fr8 Protocol store a cryptographic hash of their data on the blockchain. HoldsSensorData.sol Interface

```
/**
  * HoldsSensorData Class Interface
  */

contract HoldsSensorData {
  struct SensorDataPair {
    string dataCollectionRef;
    bytes32 dataCollectionHash;
  }

  mapping (address => bool) public sensorDataUploaders;
  SensorDataPair[] public dataCollections;

  modifier onlySensorDataUploader();

  function addSensorDataUploaders(address[] _uploaders) public;
  function removeSensorDataUploaders(address[] _uploaders) public;

  function addDataCollection(string _dataCollectionRef,
                             bytes32 _dataCollectionHash)
                             public onlySensorDataUploader;
}
```

Listing 7: HoldsSensorData.sol Interface

The HoldsSensorData class allows objects on Fr8 Protocol to store time-series sensor data, such as accelerometer data, gps location, or temperature. The data itself is stored off-chain for privacy, security, and performance. The data's unique identifier and SHA-256 hash is stored on-chain.

5 Permissions and ID Layer

On Fr8 Protocol, every user and entity is associated with public key, also known as an address, on a blockchain. Private and sensitive user information is stored off-chain, but there is always a public key associated with a user. Permissions for a shipment are stored on-chain. This allows provides an auditable access log for every component and stage of a shipment.

All incoming requests pass through the Permissions and ID Layer. Once an user can be identified by their public key, we must then authenticate them, or verify that the user is indeed who they say they are. Fr8 Protocol authenticates using

random phrases and asymmetric cryptography. This process is described in more detail below. After an user is identified and authenticated, a signed JSON Web Token(JWT) is issued. This token will be used to authorize requests until it expires. The JWT token must be attached in the header of of each request. Once a token is validated and the user is identified, the user's permissions must be retrieved. The object that the user is attempting to access is also identified by a blockchain address. Fr8 Protocol reads the permissions on that object from the blockchain and confirms the user is in fact permitted to perform the operation specified in the request. If at any point, an error or malicious attempt is detect, the request is rejected.

As previously mentioned, Fr8 Protocol uses random phrases and asymmetric cryptography to authenticate a user. The process works as follows:

1. The client calls GET /api/auth function with the user's public key in the query string.
2. The API server returns a seed.
3. The client signs this string with their private key.
4. The client POSTs to /api/auth with their public key and signed message in the body.
5. The API server verifies the message was signed correctly. If signed correctly, the API server returns a valid JWT token to the client.
6. The client makes requests with the JWT token in the Authorization Header of the request.

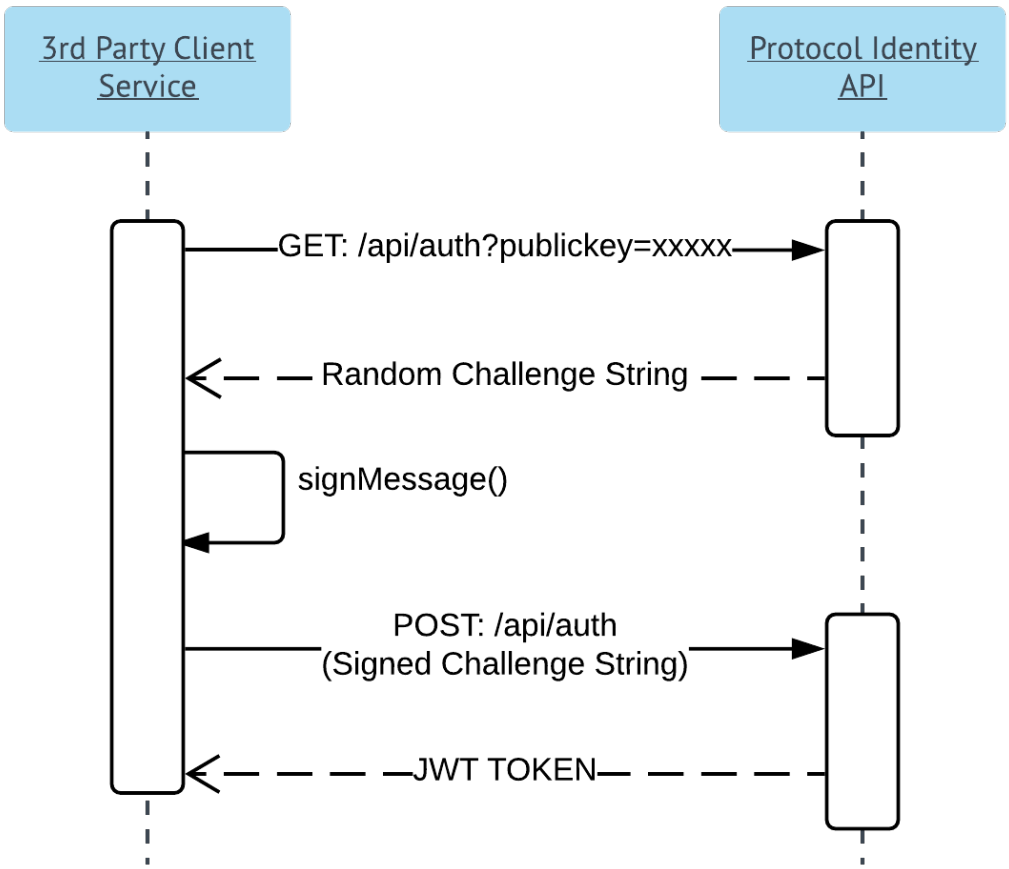

Figure 5: Fr8 Protocol Authentication

6 Interfaces

6.1 Document Interface

The Document Interface handles the creation and management of shipments, pallets, documentation, and file attachments for documentation through a Fr8 Protocol API. As previously mentioned, the object data and file attachments will be store off-chain, and the the SHA-256 hash of object data and attachment byte-code will be

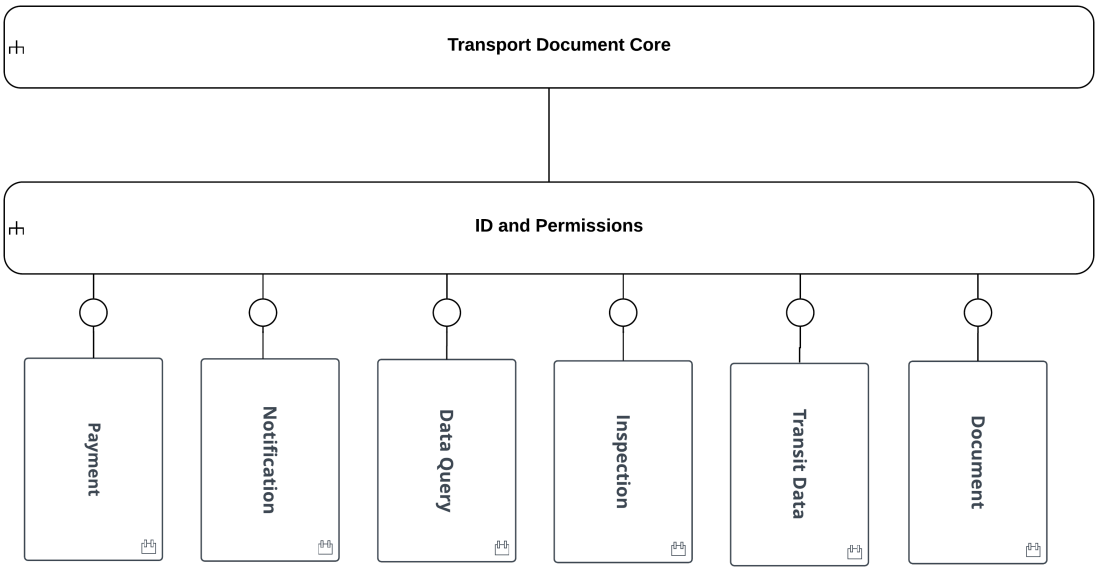

Figure 6: Fr8 Protocol Interfaces

stored on-chain. Object supports both JSON and plain-text. Image and PDF file attachments are supported. Any requests to access or modify data and attachments are corroborated against the user permissions set for a particular object. Fr8 Protocol Tokens are required to create shipments, create pallets, upload data, and upload attachments. Here is how a client application would upload a document to the Fr8 Protocol and associate it with a shipment:

1. The client deploys a Document smart contract and the contract address is returned.

2. We assume the user(inspector) has authenticated and has a valid JWT.

3. Client POSTs document to the Document interface. JSON data and optional file attachments can be submitted. The address of the associated

4. Shipment smart contract is included in the body of this request.

5. The user's Shipment permissions are validated with the permissions stored on-chain. If user is not permissioned, the Document creation request will fail.

6. Client updates Document smart contract dataHash property with SHA-256 hash of the flattened Document data and optional attachments.

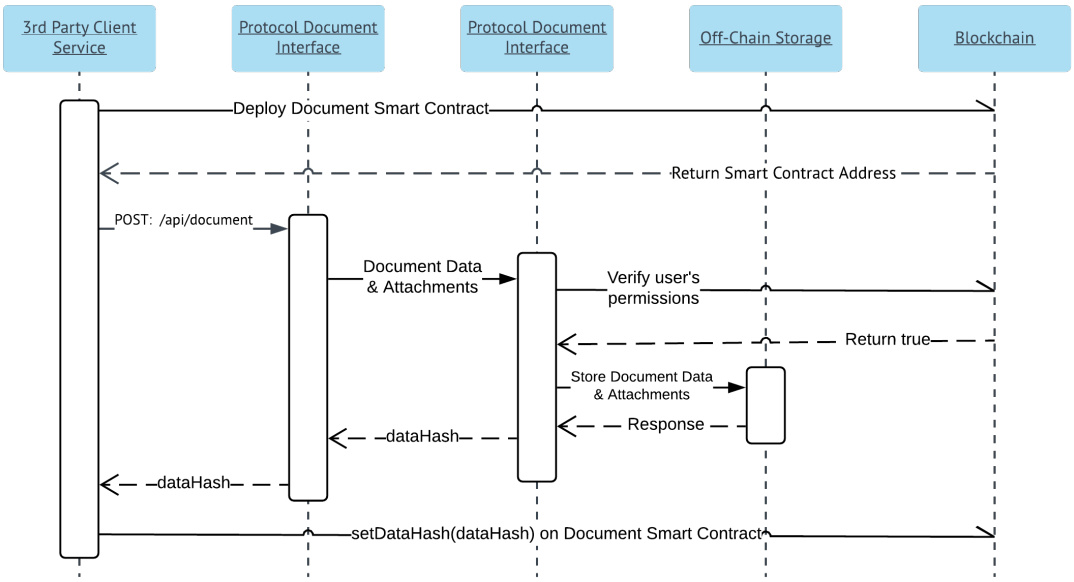

Figure 7: Fr8 Protocol Document Upload

6.2 Transit Data Interface

The Transit Data Interface is responsible for the upload of time-series sensor data of a shipment or pallet. The data will be upload via REST API in standard JSON format, where the data points themselves are contained in a JSON array, bundled in a TransitDataCollection. Transit data can only be uploaded by authorized users and devices, as defined in an object's permissions. After the data points are uploaded, the SHA-256 hash of the data is stored on-chain. Let's look at how an IoT Device would upload accelerometer data to the Fr8 Protocol:

1. We assume the IoT device has authenticated and obtained a valid JWT.

2. The IoT Devices POSTs a TransitDataCollection containing the accelerometer data to the Transit Data Interface. The address of the associated Shipment smart contract is included in this request.

3. The user's Shipment permissions are validated with the sensorDataUploader permissions stored on-chain. If user is not permissioned, the Transit Data Upload request will fail.

4. Devices calls the addDataCollection() function on the Shipment Smart Contract with the SHA-256 hash of the Transit Data and a reference to the data's off-chain storage.

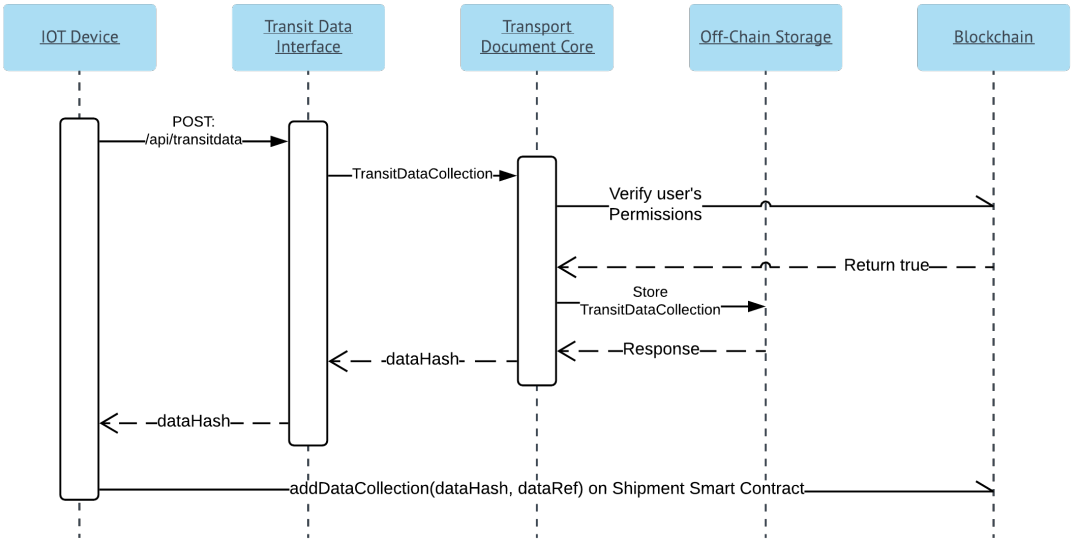

Figure 8: Fr8 Protocol Transit Data Upload

6.3 Inspection Interface

The Inspection Interface handles any inspections that a shipment or pallet may be required to pass. This interface exposes an API for the creation and management of inspection documents and file attachments. User and object permissions are checked on each request. Inspection data can be a JSON Object or plain-text. Image and PDF file attachments are supported. The inspection data and file attachments will be stored off-chain, and the SHA-256 hash of inspection data and attachment bytecode will be stored on-chain. Tokens will be required to upload inspections and

attachments. This is how a Inspection client application would upload a complete inspection to the Fr8 Protocol:

1. The client application deploys an Inspection smart contract and the contract address is returned.

2. We assume the inspector has authenticated and has a valid JWT.

3. Client POSTs inspection to the Inspection interface. JSON data and file attachments can be submitted. The address of the associated Shipment smart contract is included in the body of this request.

4. The user's permissions are validated with the inspector permissions stored on-chain. If user is not permissioned, the Inspection upload request will fail.

5. Client updates Inspection smart contract dataHash property with SHA-256 hash of the flattened Document data and attachments by calling the setDataHash() function.

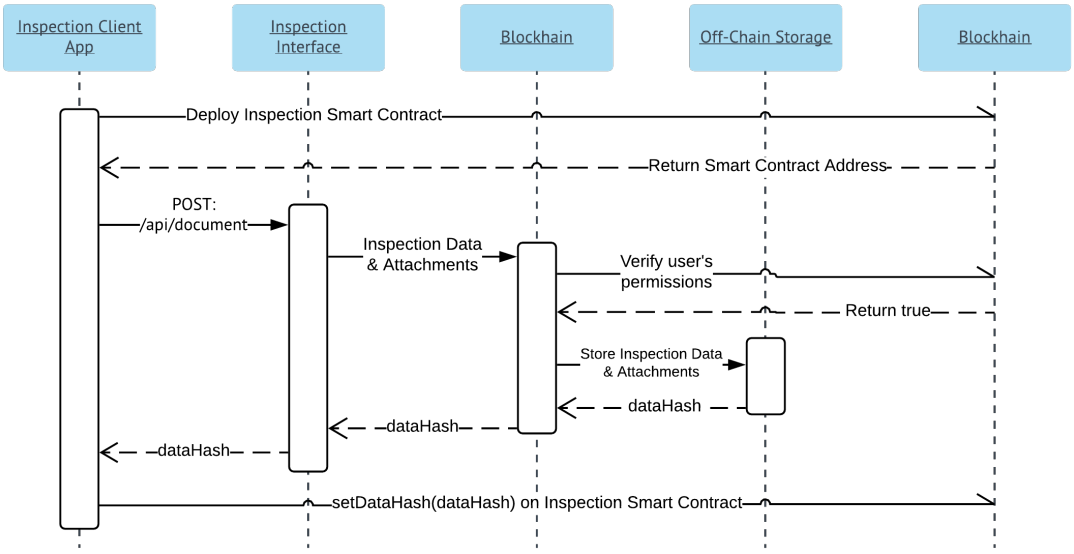

Figure 9: Fr8 Protocol Inspection Interface

6.4 Data Query Interface

The Data Query Interface provides a REST API to retrieve transit and document data. A query in Fr8 Protocol Query Language (Fr8QL) must be posted to the API. A structured JSON array will be returned, typically a collection of time-series data points. This interface supports batched and on-demand queries. Complex queries may require tokens. Here is how a user would run a query to get all of the Pallets associated to a Shipment:

1. We assume the user has authenticated and has a valid JWT.
2. The users POSTs a string query to the Data Query Interface. The query could look like this:

```
/api/query/pallets?q={ "Shipments.contractAddress": "0
    x0e320b36b45c84f7522a1424ba3785f07ead90ba" }
```

3. The user's permissions for accessing any objects involved in the query are validated.
4. The query is executed and the results are returned.

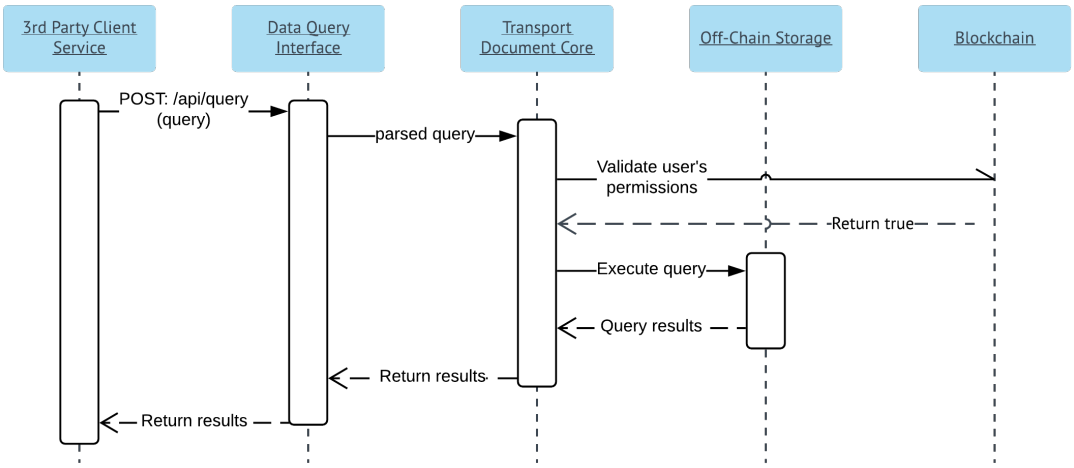

Figure 10: Fr8 Protocol Query Interface

6.5 Payment Interface

The Payment Interface emits payment information and events to webhooks. The webhook endpoints and contents are both configurable in the Shipment Document. A log of every emitted payment event is stored on chain. The Payment Interface webhook endpoints are connected to a Payment Service. After a payment event is emitted, the Payment Interface is expecting a corresponding PaymentReceipt object to be posted to it via a REST API. The contents of the PaymentReceipt will be stored off chain, but its SHA-256 hash will be stored on-chain alongside the associated shipment. Let's take a look at how the Payment Interface operates when it is configured to send a payment once a Shipment has been marked as Received on the Fr8 Protocol:

1. A Shipment on the Fr8 Protocol is updated and marked as Received when it arrives to a distribution center.

2. Upon this update, the Transport Document Core reads the Shipment's configuration and sees that this Shipment is configured to fire off a payment event upon arrival at this distribution center. The configuration also holds information about which Payment Service to use, the webhooks to be pinged, the recipient, the payment amount, and other relevant data.

3. The Payment Interface sends an HTTP request to the specified Payment Service.

4. The Payment Service processes the request and POSTs a PaymentReceipt back to the Payment Interface. The PaymentReceipt contains information relating to the result of the payment, such a transaction id or status.

5. Once the Payment Interface receives the PaymentReceipt, it stores the PaymentReceipt's contents off-chain. The SHA-256 hash of all the PaymentReceipts is calculated and stored on-chain by calling the setPaymentReceiptsHash() function on the Shipment smart contract.

6.6 Notification Interface

The Notification Interface emits notification information and events to webhooks. The webhook endpoints and contents are both configurable in the Shipment Document. A log of every emitted notification event is stored on-chain. Notification

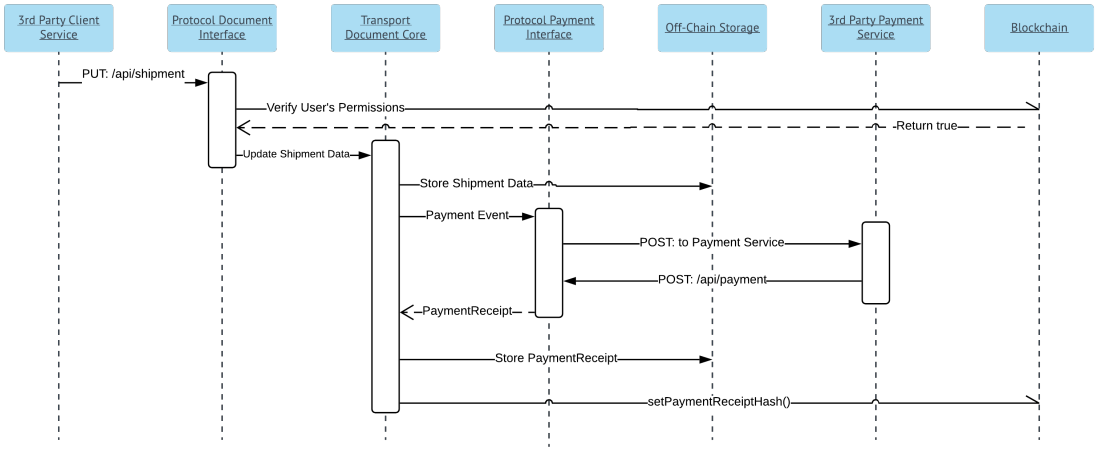

Figure 11: Fr8 Protocol Payment Interface

Events and Triggers are configured in Shipment Document as well. The Notification Interface webhook endpoints are connected to a Notification Service. After a notification event is emitted, the Notification Interface is expecting a corresponding Notification Receipt object to be posted to it via a REST API. The contents of the Notification Receipt will be stored off-chain, but its SHA-256 hash will be stored on-chain alongside the associated shipment. Let's take a look at how the Notification Interface operates when it is configured to send a notification once a Shipment has been marked as Received on the Fr8 Protocol:

1. A Shipment on the Fr8 Protocol is updated and marked as Received when it arrives to a distribution center.

2. Upon this update, the Transport Document Core reads the Shipment's configuration and sees that this Shipment is configured to fire off a notification event upon arrival at this distribution center. The configuration also holds information about which Notification Service to use, the webhooks to be pinged, the recipient, and the message contents.

3. The Notification Interface sends an HTTP request to the specified Notification Service.

4. The Notification Service processes the request and POSTs a NotificationReceipt back to the Notification Interface. The NotificationReceipt contains in-

formation relating to the result of the notification, such a an id or status.

5. Once the Notification Interface receives the NotificationReceipt, it stores the NotificationReceipt's contents off-chain. The SHA-256 hash of all the NotificationReceipts is calculated and stored on-chain by calling the setNotificationReceiptsHash() function on the Shipment smart contract.

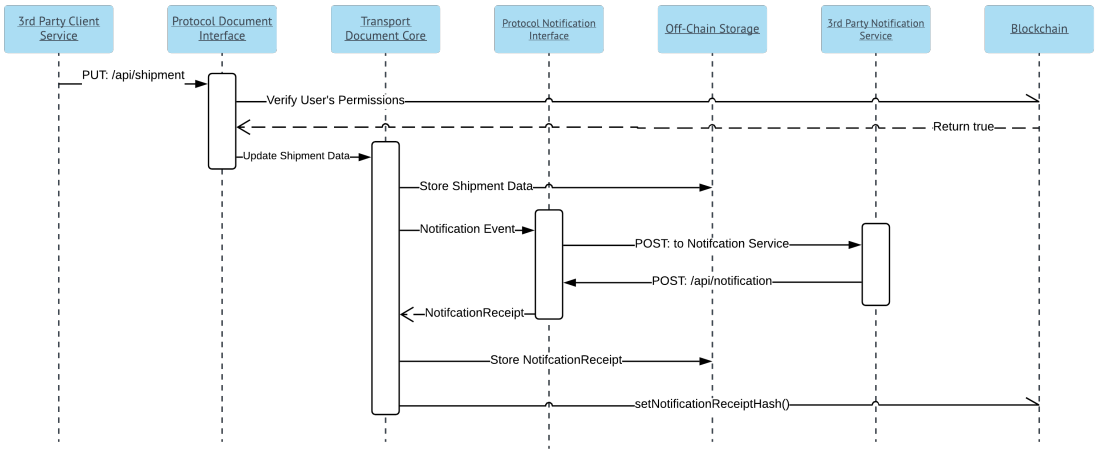

Figure 12: Fr8 Protocol Notification Interface

7 Conclusion

Fr8 Protocol is designed and architected for simplicity, modularity, and interoperability. Design decisions were made to empower rapid adoption and simple onboarding. For example, businesses are not forced to change their current processes or adopt new standards. Expansion of capabilities for the protocol are achieved through direct improvements and feature-rich add-ons supported through the services layer.

Services built on top of the protocol are almost limitless. Applications which interact with the services include but aren't limited to Track and Trace, Document Management, IoT System Integrations, Data visualization and analytics, CRM and legacy ERP connectivity.

Dissemination of Fr8 Protocol will have myriad benefits: significant reduction of EDI, shared single version of the truth, multi-party Master Data, fast development of

new applications, and unparalleled business agility with default performance, security and interoperability.