5

Conflict Resolution/Non-repudiation Protocol

5.1 Objectives

The conflict resolution system's main goal is to prevent and/or solve conflicts when invoicing for a given exchange of data. It is therefore a core subsystem for the i3-MARKET secure data exchanges.

For the conflict resolution system to work, the i3-MARKET Non-repudiation Protocol (NRP) must be executed with every exchanged block of data. The Non-repudiation Protocol generates verifiable proofs of the data exchange that can be used to later prove that a given digital data exchange happened and that it met the agreed conditions (based on a data sharing agreement).

If the NRP is followed, the NRP proofs can be used to support fair unfakeable billing with fiat or crypto money and to prevent or solve eventual disputes with the data exchange alike.

A complementary conflict-resolver service (CRS) has been developed, which can be run by any trusted third party to issue verifiable signed resolutions regarding the execution of the NRP.

In short, as per the above explanation, the conflict resolution/Non-repudiation Protocol system relies on two subsystems, both already made publicly available in the i3-MARKET GitHub and Gitlab repo:

- the Non-repudiation Protocol library [60];
- the conflict-resolver service [61].

Updated detailed documentation can be found in, e.g., https://github.com/i3-Market-V3-Public-Repository/SP3-SCGBSSW-CR-Documentation#conflict-resolution--non-repudiation-protocol.

5.2 Technical Requirements

The conflict resolution/Non-repudiation Protocol must prevent the following situations between the two peers of a data exchange, namely provider and consumer:

- to deny that a given data-block exchange happened;
- or to assert that a data-block exchange that did not happen, happened.

As a result, providers will not be able to invoice a consumer for a data-block not exchanged; and consumers will not be able to deny or cancel a payment for a data-block that was successfully exchanged.

For it to happen, every block of data must be exchanged using the NRP. Accounted proofs give no room to alter the invoicing (fiat money) or the crypto payments (i3-MARKET tokens) if both entities reliably execute the protocol; otherwise, the conflict resolver service can be invoked to univocally solve which entity is intentionally or unintentionally malfunctioning.

5.3 Solution Design/Blocks

The Non-repudiation Protocol starts with a provider Alice, hereby A, sending a signed proof of origin (PoO) along with an encrypted block of data to a consumer Bob, hereby B.

An overview of the protocol is depicted in Figure 5.1, and more detailed sequence diagrams of every step are provided in the following sections.

After validating the PoO, B will demonstrate his will to get the data by sending a signed proof of reception (PoR). Just recall that B is at this point not yet able to decrypt the data since he does not know the secret to decrypt them.

The PoR is a proof that can be used by A to demonstrate that B is committed to get the secret to decrypt the block of data.

Now A can release the secret as part of a proof of publication (PoP). However, as B may state that he did not receive the PoP, A also publishes the secret to the ledger. It is now under B's responsibility to get the secret from the ledger since he implicitly agreed to it when sending the PoR.

For A to create a valid invoice for that block of data, she *must* present a valid PoR and demonstrate that the secret was published to the ledger within the agreed delay (part of the agreement). As a result, the lack of one or both proofs will result in an invalid invoice.

The conflict-resolver service (CRS) can be queried to provide a signed resolution about the Non-repudiation Protocol associated with an invoice

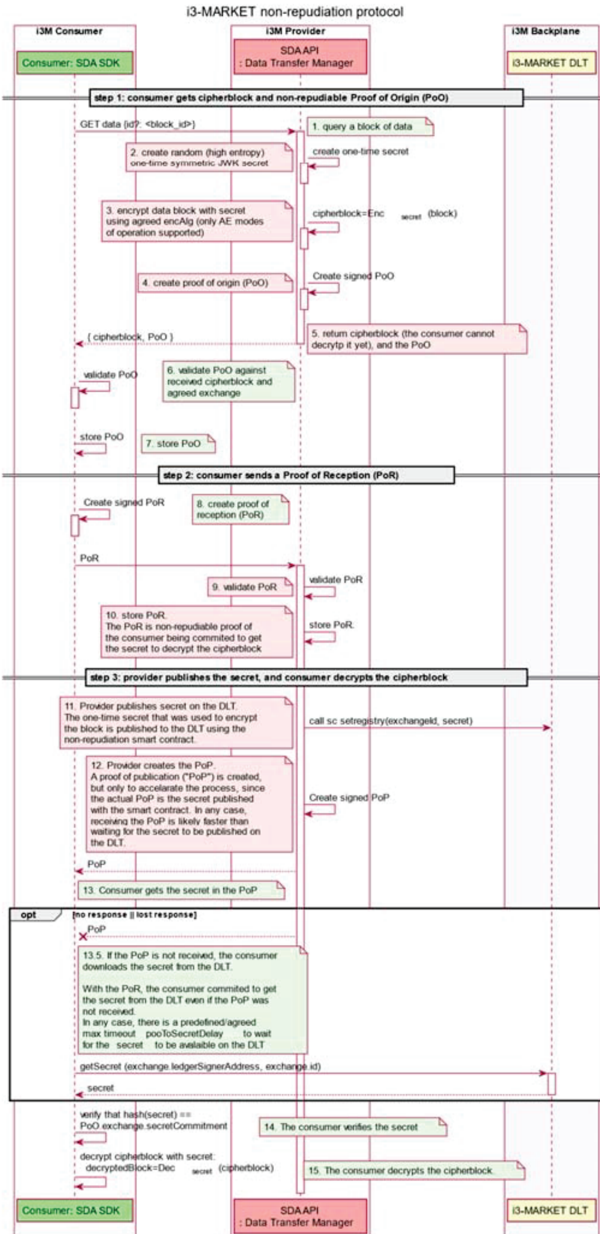

Figure 5.1 Overview of the Non-repudiation Protocol.

being valid or invalid. It could be invoked by either the consumer or the provider. The latter should be mandatory, being the resolution sent along with the invoice to the consumer.

However, this resolution does not ensure that the published secret could be used to decrypt the encrypted block of data. If the consumer B is not able to decrypt the cipherblock, he could initiate a dispute on the CRS. The CRS will also provide signed resolution of whether B is right or not.

5.4 Diagrams

This section presents detailed diagrams for the Non-repudiation Protocol and conflict resolution already depicted in the previous section. For a diagram with high-level overview of the NRP, please refer to Figures 5.2–5.6 for the different interactions and use cases.

- **NRP — step 1: consumer gets cipherblock and non-repudiable proof of origin (PoO).**
- **NRP — step 2: consumer sends a proof of reception (PoR).**
- **NRP — step 3: provider publishes the secret, and consumer decrypts the cipherblock.**
- **Conflict resolution: verification (NRP completeness).**
- **Conflict resolution: dispute.**

5.5 Interfaces

A standard i3-MARKET developer interacts with the conflict resolution/Non-repudiation Protocol system using the API of the non-repudiation library from the JavScript/TypeScript code or querying the conflict resolver service HTTP API.

API of the non-repudiation library:

The non-repudiation library API is a documented typescript library whose API can be properly documented "on the fly" while programming. Besides that, automated TypeDoc documentation is generated and available at https://github.com/i3-Market-V3-Public-Repository/SP3-SCGBSSW-CR-NonRepudiationLibrary/blob/public/docs/API.md.

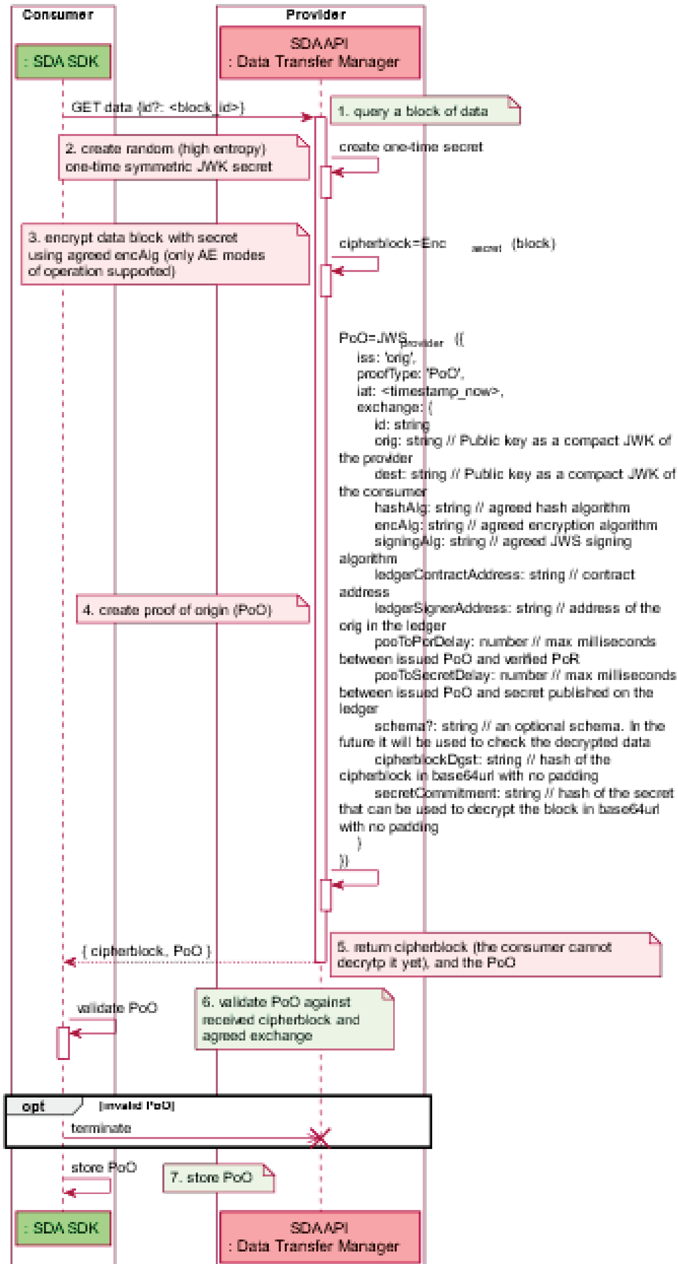

Figure 5.2 NRP — step 1: consumer gets cipherblock and non-repudiable proof of origin (PoO).

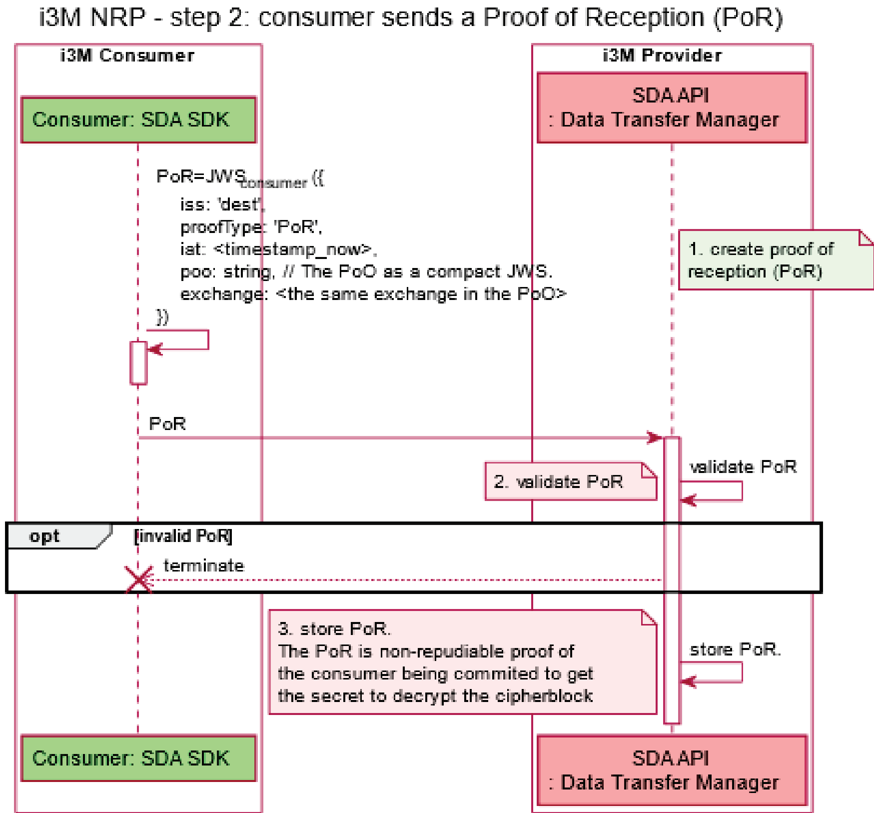

Figure 5.3 NRP — step 2: consumer sends a proof of reception (PoR).

API of the conflict resolver service:

The conflict resolver service implements a HTTP API following the OpenAPI standard. The specification can be consulted in the openapi.json file in the root directory of the conflict resolver service at [61]. For convenience, it can also be visualized online at editor.swagger.io[1] as it is shown in Figure 5.7.

The CRS provides two endpoints: one for checking that the protocol was executed properly, and the other one to initiate a dispute when a consumer claims that he cannot decrypt the cipherblock he has been invoiced for.

[1] https://editor.swagger.io/?url=https://raw.githubusercontent.com/i3-MARKET-V3-Public-Repository/SP3-SCGBSSW-CR-ConflictResolverService/public/spec/openapi.yaml

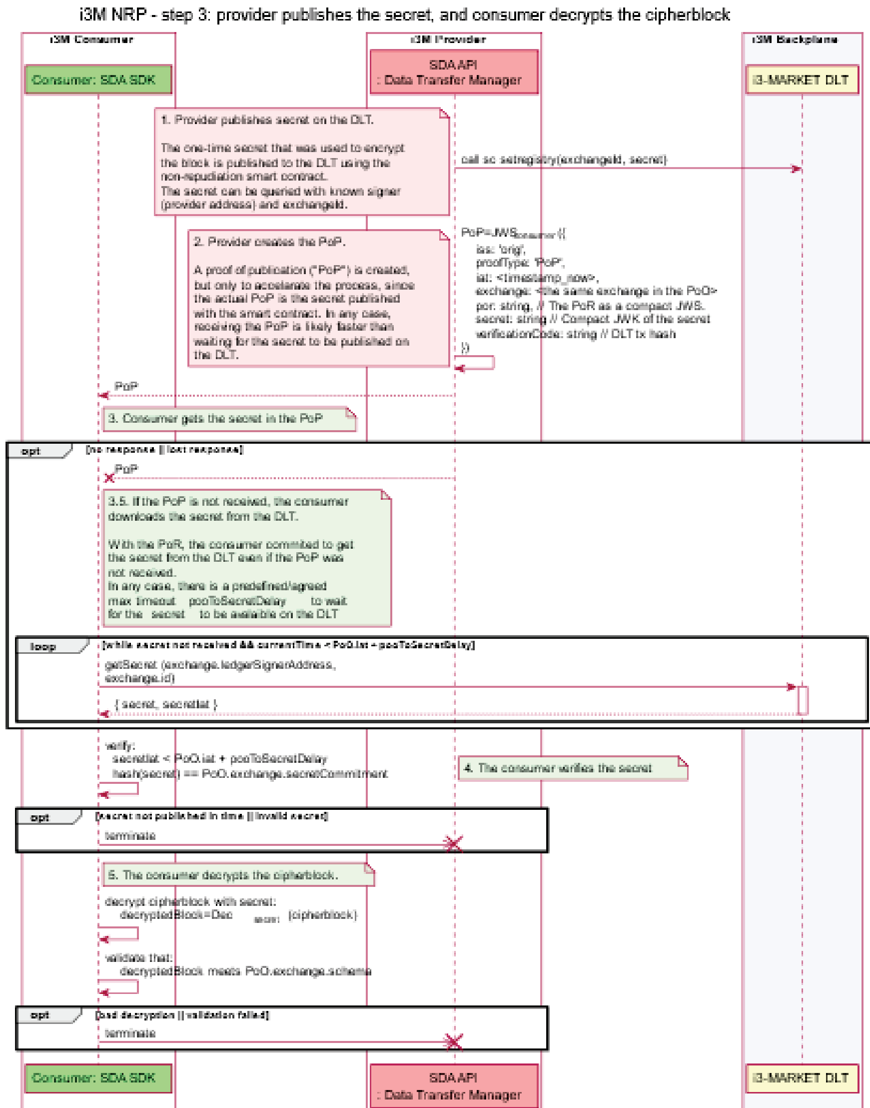

Figure 5.4 NRP — step 3: provider publishes the secret, and consumer decrypts the cipherblock.

The endpoints require JWT bearer authentication. The JWT can be obtained after performing a login with OIDC and presenting valid i3-MARKET credentials.

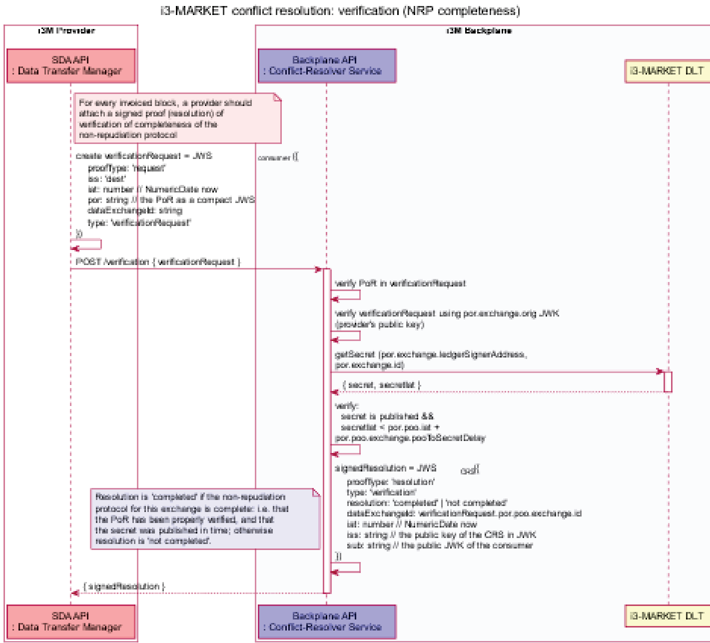

Figure 5.5 Conflict resolution: verification (NRP completeness).

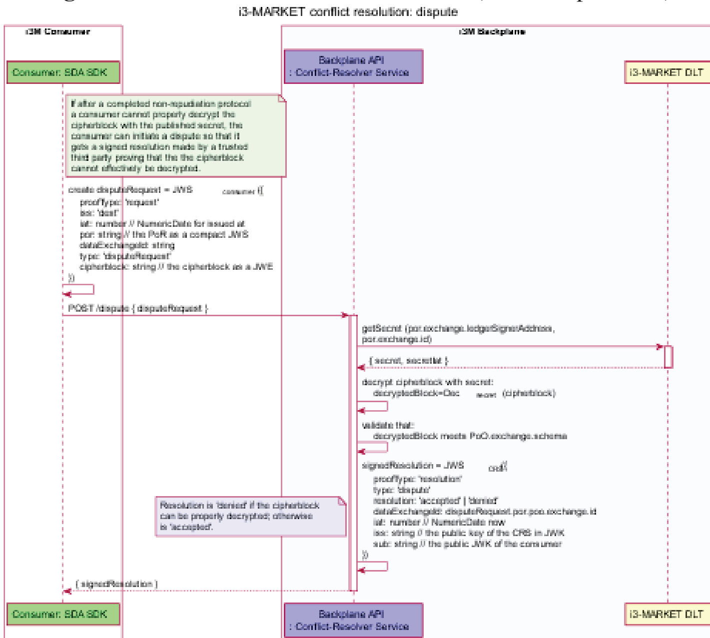

Figure 5.6 Conflict resolution: dispute.

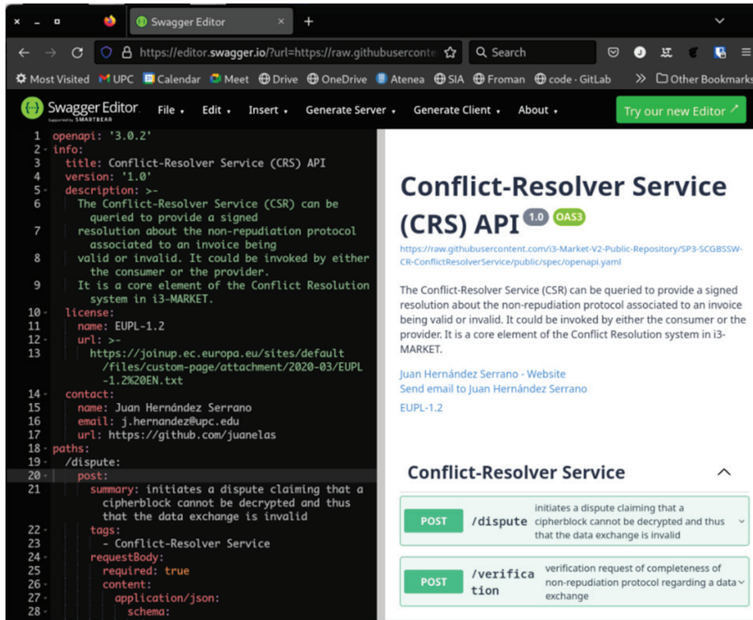

Figure 5.7 CRS API at swagger.editor.io.

- **POST/verification.**

The CRS can be queried to provide a signed resolution about a data exchanged successfully performed or not. It could be invoked by either the consumer or the provider. The provider should query this endpoint and send it along with the invoice to the consumer.

This endpoint can be accessed at POST/verification and requires valid i3-MARKET consumer or provider's credentials.

Input:

A verification request as a compact JSON Web Signature (JWS). For the request to be accepted, it *must* be signed with the same key it was used during the data exchange for this verification.

```
{
  verificationRequest: string // the verification request in compact JWS format
}
```

A verification request is a JWS signed by either the consumer or the provider using the same key he/she used for the data exchange. The verification request payload holds a valid PoR:

```
{
  type: 'verificationRequest'
  proofType: 'request'
  iss: 'orig' | 'dest'
  iat: number // unix timestamp for issued at
  por: string // a compact JWS holding a PoR. The proof MUST be signed with the same key as either 'orig' or 'dest' of the payload proof.
  dataExchangeId: string // the unique id of this data exchange
}
```

Output:
It returns a signed resolution as a compact JWS with payload:

```
{
  proofType: 'resolution'
  type: 'verification'
  resolution: 'completed' | 'not completed' // whether the data exchange has been verified to be complete
  dataExchangeId: string // the unique id of this data exchange
  iat: number // unix timestamp stating when it was resolved
  iss: string // the public key of the CRS in JWK
  sub: string // the public key (JWK) of the entity that requested a resolution
}
```

- **POST/dispute.**

Note that the signed resolution obtained from POST/verification does not ensure that the published secret could be used to decrypt the encrypted block of data. If the consumer B is not able to decrypt the cipherblock, he could initiate a dispute on the CRS. The CRS will also provide signed resolution of whether B is right or not.

All this is handled in this endpoint, which can only be queried if in possession of valid i3-MARKET consumer's credentials.

Input:

```
{
  disputeRequest: string // the dispute request in compact JWS format
}
```

A dispute request as a compact JSON Web Signature (JWS). For the request to be accepted, it *must* be signed with the same key it was used during the data exchange for this verification.

The payload of a decoded disputeRequest holds a valid PoR, and the received cipherblock:

```
{
  proofType: 'request'
  type: 'disputeRequest'
  iss: 'dest'
  cipherblock: string // the cipherblock as a JWE string
  iat: number // unix timestamp for issued at
  por: string // a compact JWS holding a PoR. The proof MUST be signed with the
same key as either 'orig' or 'dest' of the payload proof.
  dataExchangeId: string // the unique id of this data exchange
}
```

Output:
It returns a signed resolution as a compact JWS with payload:

```
{
  proofType: 'resolution'
  type: 'dispute'
  resolution: 'accepted' | 'denied' // resolution is 'denied' if the cipherblock
can be properly decrypted; otherwise is 'accepted'
  dataExchangeId: string // the unique id of this data exchange
  iat: number // unix timestamp stating when it was resolved
  iss: string // the public key of the CRS in JWK
  sub: string // the public key (JWK) of the entity that requested a resolution
}
```

5.6 Background Technologies

Both the non-repudiation library and the conflict resolver service need access to a DLT. Access to the DLT is provided by the following technologies:

- Ethers.js [55] is a complete and compact library for interacting with the Ethereum-based DLTs. Along with Web3 is the reference implementation for that purpose.
- Veramo [56] is a JavaScript Framework for Verifiable Data that was designed from the ground up to be flexible and modular, which makes it highly scalable. It can run on several environments: node, mobile, and browser. Its main utility is to make easy the use of DIDs, Verifiable Credentials, and data-centric protocols to bring next-generation features to users.

The smart contracts that regulate the Non-repudiation Protocol have been developed in Solidity [37], an object-oriented, high-level language for implementing smart contracts for Ethereum-like DLTs, and the development environment of choice has been Hardhat.

The non-repudiation library can be instantiated from JavaScript or TypeScript code. It internally uses Panva's JOSE [63] to handle JSON web keys, and Ajv [64] to check and verify JSON schema.

Conflict resolver service HTTP API is developed using Express [65], a minimal and flexible Node.js web application framework that provides a robust set of features for creating robust APIs (among other things).

The conflict resolver service meets the OpenAPI specification [57] with validation of all inputs against the OpenAPI schema using express-openapi-validator [58].