7

Smart Contract Manager

7.1 Objectives

The smart contract manager (SCM) provides a gateway to access the smart contracts and is used by other subsystems to integrate their functionalities (conflict resolution, pricing manager, explicit user consent, and secure data exchanges).

Smart contract manager facilitates the creation of agreement objects using the data sharing agreement (DSA) smart contract. The DSA solidity contract is based on a legal agreement for data sharing, considering the existing legal framework (e.g., GDPR [26]). The agreement objects are used to enforce agreed-upon obligations from the provider and consumer sides.

The smart contract manager development has been made publicly available in the i3-MARKET GitHub repository and the smart contracts the subsystem uses at [66]. The Table 7.1 summarizes the Smart Contract Manager user stories.

7.2 Technical Requirements

Table 7.1 Smart contract manager – user stories.

Name	*Description*	*Labels*
SCM	Within i3-MARKET, DSA objects need to be stored on the blockchain in order to automatically enforce certain clauses of the legal data trading agreement. Additionally, automatic conflict resolution of certain types of violations has to be supported. The smart contracts of the SCM need to combine legal certainty with automated enforcement, built-in conflict resolution mechanisms, and guaranteed access to remedy. The SCM evaluates a signed resolution, issued by the conflict-resolver service, which relies on the execution of the Non-repudiation Protocol. Depending on the type of resolution, the state of the agreement is automatically updated.	User story

Table 7.1 *Continued.*

Name	*Description*	*Labels*
	Explicit data owner consent: In case of personal data, legal consent of data owners is required. When the consent is given, the SCM stores a list of explicit consents for a specific offering. The consent can be revoked anytime, and before an agreement is created, the consent status is verified. As long as the data to be shared is personal data, agreements can be created just when the consent was given by the data owner. Pricing: The price and the fee of the data are stored in the agreement. The fee is requested from the pricing manager, based on the price in the data offering.	

7.3 Solution Design/Blocks

The smart contract manager extracts the contractual parameters from the data offering description and returns a template with possible contractual parameters (to be displayed in the marketplace), as shown in Figure 7.1. After a data purchase request is sent, with a potential proposal of new parameters by the consumer, the provider and consumer must sign the agreement and store it in the wallet. As soon as both received the signed data sharing agreement and saved it in the wallet, the provider can create and store the agreement on the blockchain. The smart contract manager invokes the data sharing agreement smart contract and creates an agreement with the proposed contractual parameters. The agreement object is put on the ledger and automatically enforced by the corresponding smart contract (Figure 7.2).

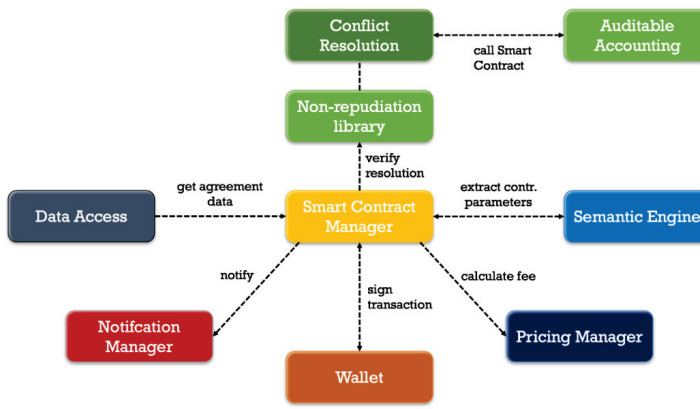


Figure 7.1 Context view of the smart contract manager.

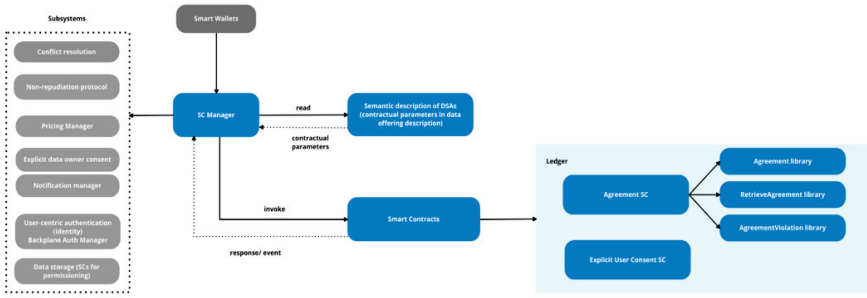


Figure 7.2 Component diagram of the smart contract manager subsystem.

The smart contract manager is interconnected with the following i3-MARKET subsystems, as it is shown in Figures 7.3–7.5.

- **Semantic engine:** To retrieve the parameters and details about the data offering descriptions to compile information for the contract agreements.
- **Conflict resolution:** In order to check whether a violation to the contract occurred, the conflict resolution is invoked. The conflict resolution will prevent any 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. The conflict-resolver service issues verifiable signed resolutions regarding the execution of the i3-MARKET Non-Repudiation Protocol. The SCM evaluates the signed resolution and, depending on the type of resolution, automatically changes the state of the agreement in case of a violation, as well as suggests penalties for one of the peers.
- **Non-repudiation Protocol:** The Non-repudiation Protocol aims at preventing parties in a data exchange from falsely denying having taken part in that exchange.
- **Explicit data-owner consent:** To ensure an explicit consent of the data owners every time their personal data is traded, the explicit data owner consent component is triggered.
- **Pricing manager:** The SCM requests the fee of the data based on the price registered in the data offering by invoking the pricing manager to calculate the corresponding fee and includes it in the contractual template.
- **User-centric authentication:** To ensure that only authorized participants (with the corresponding role) are able to trigger functionality

provided by the data sharing agreement smart contract (via the smart contract manager), user-centric authentication is used (part of the Backplane).

- **i3M-Wallet:** The raw transactions created in the SCM have to be signed with an i3M-Wallet (either the Wallet Desktop App or the server wallet) in order to deploy them.

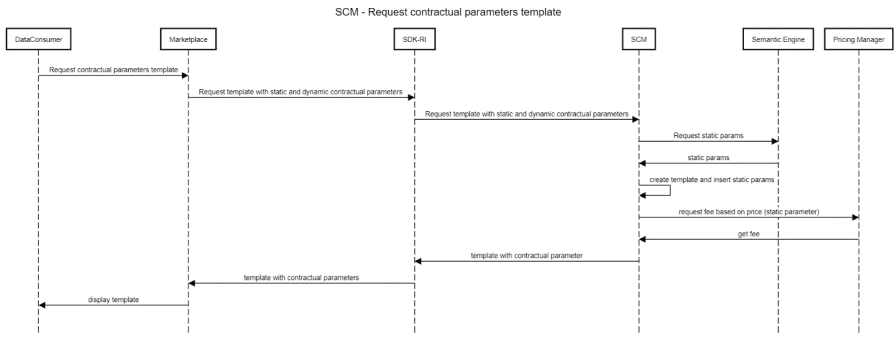


Figure 7.3 Sequence diagram – retrieve contractual parameters template.

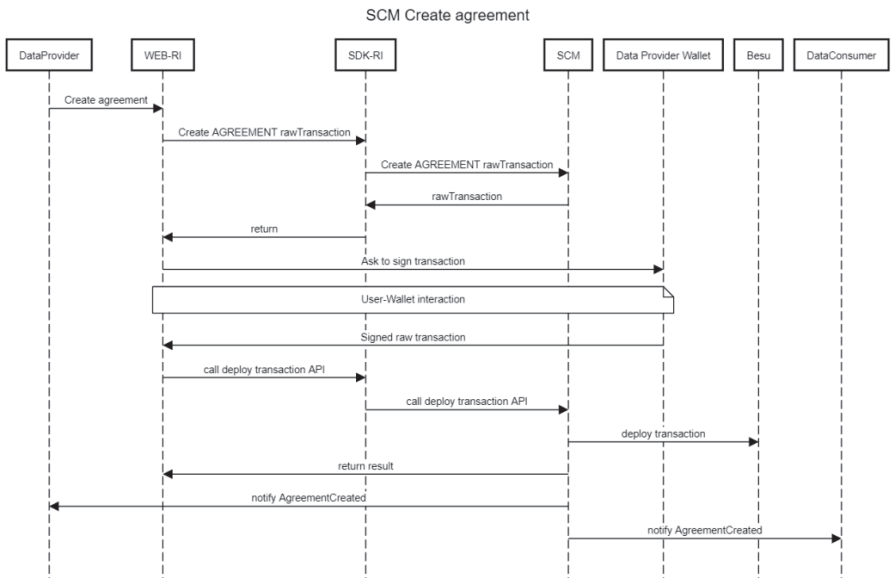


Figure 7.4 Sequence diagram – create agreement.

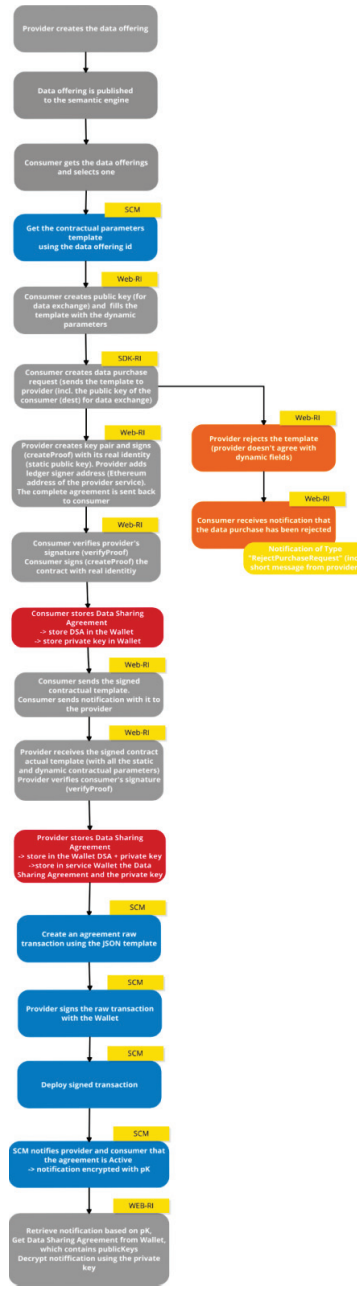


Figure 7.5 Data sharing agreement negotiation, key pair generation, storage in wallet, and agreement creation on blockchain.

7.4 Diagrams

The smart contract manager extracts the static contractual parameters from the data offering description using the semantic data model. The interactions are shown in Figure 7.6. The dynamic parameters, such as the consumer DID, start date, and end date of the agreement, are filled when a data purchase request is created by the consumer.

Before storing an agreement on the blockchain using the smart contract manager, the provider and the consumer should generate their public−private keys (using the non-repudiation library) and they should each sign the contract. After they filled in their public keys and the contract is signed, they should store the generated key pairs and data sharing agreement in their wallets as shown in Figure 7.7.

As soon as the negotiation between the provider and consumer is over and they agree on specific contractual parameters, as well as store the final data sharing agreement and the key pairs in their wallets, the provider can create the agreement on the blockchain using the smart contract manager.

Firstly, a raw transaction is created using the data sharing agreement, which was saved in the wallet. The successful response of creating an agreement request is a raw transaction object. This raw transaction has to be signed with the wallet using the provider's DID. After the signed transaction is obtained from the wallet, it has to be deployed. The response of the Smart Contract Manager should be a transaction object with information about

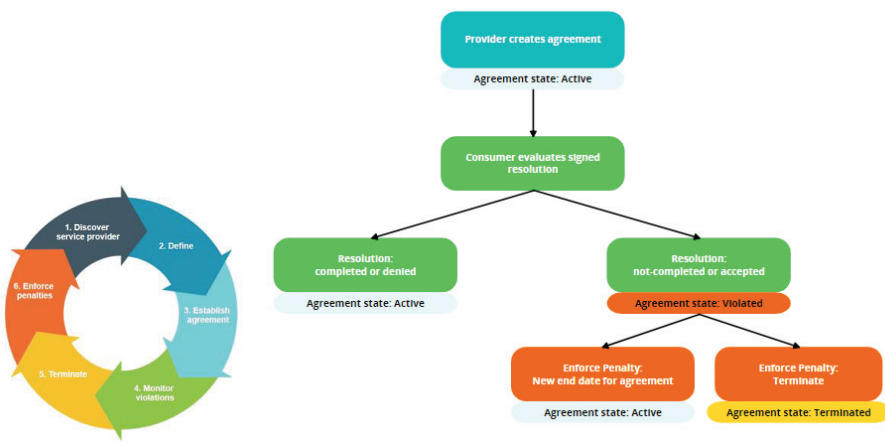


Figure 7.6 Sequence diagram — check agreements by offering ID.

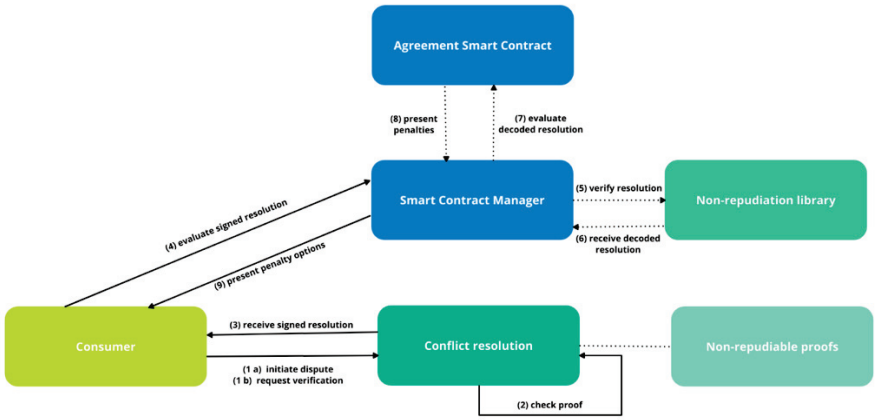


Figure 7.7 Conflict resolution.

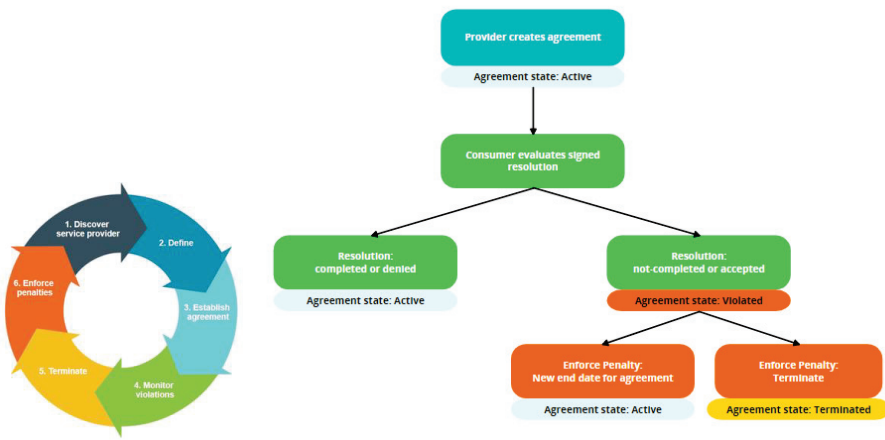


Figure 7.8 Agreement lifecycle and states.

the transaction in Figure 7.8. If the confirmation is 1, the transaction was successfully deployed, and the agreement is stored on the blockchain.

After that, the provider and consumer receive a notification that the agreement is active, which means it was created and stored on the blockchain. This notification will be encrypted and contains the agreement id. The notifications should be retrieved from the notification manager based on the provider/consumer public key and decrypted using the corresponding private key. After they receive this notification, the provider should *post* the data

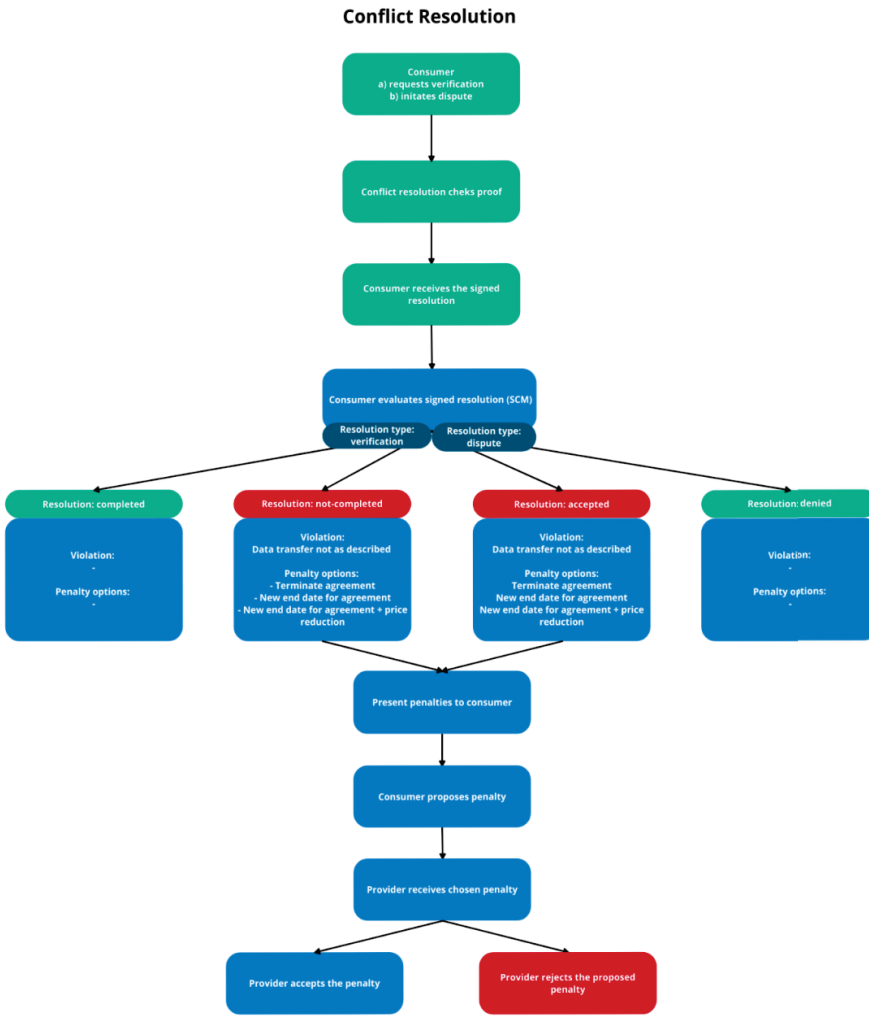


Figure 7.9 Agreement violation — conflict resolution.

exchange agreement, the agreement id, and the private key to data access and then the consumer can start the transfer – see Figure 7.9.

Agreement violation — conflict resolution:
After the data transfer is finished, a consumer can request a verification or initiate a dispute using the conflict resolution. The proof of the completeness

of the data exchange will be checked and the consumer receives the signed resolution based on that proof.

The smart contract manager evaluates the signed resolution. Within this evaluation, the resolution is decoded and depending on the resolution, the agreement's state can change from active to violated.

The transfer was unsuccessful when the resolution is:

- not completed (in case of a verification) – the decryption key was not published;
- accepted (in case of a dispute) – the cypher block cannot be properly decrypted.

If the transfer was not successful, the agreement is violated. When the agreement is violated, the consumer receives a list of penalties.

These penalties could be:

- new end date for agreement;
- new end date for agreement and a price reduction;
- termination of agreement.

The consumer should propose one of these penalties to the provider. The provider will receive a notification with the chosen penalty and if he agrees to the penalty, he should enforce on the blockchain. By enforcing the new penalty, the agreement state changes from violated to active or terminated (in case the penalty termination is chosen).

7.5 Interfaces

The smart contract manager API is the interface via which the clients gain access to the smart contract parameters.

The endpoints documented below were grouped by modules.

Agreement:

```
GET /template/{offering_id}
```

Request template with static and dynamic parameters

offering_id (required)

Example data

Content-Type: application/json

```
{
  "dataOfferingDescription": {
    "dataOfferingId": "63662ebdb7d5dd78b7159566",
    "version": 0,
```

```
    "title": "Oil Supply Unit",
    "category": "manufacturing",
    "active": true
  },
  "parties": {
    "providerDid":
"did:ethr:i3m:0x0243cc9dbc7157ee12ce1898ac0c49b366822f32d57bc108e127f45b6c4
3a57e90",
    "consumerDid": "string"
  },
  "purpose": "Oil supply Unit measurements",
  "duration": {
    "creationDate": 0,
    "startDate": 0,
    "endDate": 0
  },
  "intendedUse": {
    "processData": true,
    "shareDataWithThirdParty": false,
    "editData": true
  },
  "licenseGrant": {
    "transferable": false,
    "exclusiveness": true,
    "paidUp": true,
    "revocable": true,
    "processing": true,
    "modifying": true,
    "analyzing": true,
    "storingData": true,
    "storingCopy": true,
    "reproducing": true,
    "distributing": false,
    "loaning": false,
    "selling": false,
    "renting": false,
    "furtherLicensing": false,
    "leasing": false
  },
  "dataStream": false,
  "personalData": false,
  "pricingModel": {
    "paymentType": "one-time purchase",
    "pricingModelName": "string",
    "basicPrice": 125.68,
    "currency": "$",
    "fee": 6.28,
    "hasPaymentOnSubscription": {
      "paymentOnSubscriptionName": "",
      "paymentType": "",
      "timeDuration": "",
      "description": "",
      "repeat": "",
      "hasSubscriptionPrice": 0
    },
    "hasFreePrice": {
      "hasPriceFree": false
    }
  },
  "dataExchangeAgreement": {
    "orig": "string",
```

```
    "dest": "string",
    "encAlg": "A128GCM",
    "signingAlg": "ES256",
    "hashAlg": "SHA-256",
    "ledgerContractAddress": "0x8d407a1722633bdd1dcf221474be7a44c05d7c2f",
    "ledgerSignerAddress":
"0x02897978ebd80646bc469cba19d79d8655cd862cb9fd2484141d66103260cc540d",
    "pooToPorDelay": 100000,
    "pooToPopDelay": 30000,
    "pooToSecretDelay": 180000
  },
  "signatures": {
    "providerSignature": "string",
    "consumerSignature": "string"
  }
}
```


Returns the template with static and dynamic contractual parameters

`POST /sdk-ri/contract/create-data-purchase`

Create data purchase request (not part of the Backplane) – sends notification to provider with the static and dynamic parameters filled in by the consumer

`POST /create_agreement_raw_transaction/{sender_address}`

Create agreement raw transaction (createAgreement)
sender_address (required)
Request body
body template (required)


```
{
  "dataOfferingDescription": {
    "dataOfferingId": "63662ebdb7d5dd78b7159566",
    "version": 0,
    "title": "Oil Supply Unit",
    "category": "manufacturing",
    "active": true
  },
  "parties": {
    "providerDid":
"did:ethr:i3m:0x0243cc9dbc7157ee12ce1898ac0c49b366822f32d57bc108e127f45b6c4
3a57e90",
    "consumerDid":
"did:ethr:i3m:0x03878572e4476a6b7b0223d07f53159ef923c874084ea56760fd130d80c
51409ad"
  },
  "purpose": "P&ID diagram of the Lube Oil supply Unit",
  "duration": {
    "creationDate": 1678997655,
    "startDate": 1786678869,
    "endDate": 1886678869
  },
  "intendedUse": {
    "processData": true,
    "shareDataWithThirdParty": false,
    "editData": true
  },
```

```
  "licenseGrant": {
    "transferable": false,
    "exclusiveness": false,
    "paidUp": true,
    "revocable": true,
    "processing": true,
    "modifying": true,
    "analyzing": true,
    "storingData": true,
    "storingCopy": true,
    "reproducing": true,
    "distributing": false,
    "loaning": false,
    "selling": false,
    "renting": false,
    "furtherLicensing": false,
    "leasing": false
  },
  "dataStream": false,
  "personalData": false,
  "pricingModel": {
    "paymentType": "one-time purchase",
    "pricingModelName": "string",
    "basicPrice": 125.68,
    "currency": "$",
    "fee": 6.28,
    "hasPaymentOnSubscription": {
      "paymentOnSubscriptionName": "string",
      "paymentType": "string",
      "timeDuration": "string",
      "description": "string",
      "repeat": "string",
      "hasSubscriptionPrice": 0
    },
    "hasFreePrice": {
      "hasPriceFree": false
    }
  },
  "dataExchangeAgreement": {
    "orig":                                    "{\"kty\":\"EC\",\"crv\":\"P-
256\",\"x\":\"4sxPPpsZomxPmPwDAsqSp94QpZ3iXP8xX4VxWCSCfms\",\"y\":\"8YI_bvV
rKPW63bGAsHgRvwXE6uj3TlnHwoQi9XaEBBE\",\"alg\":\"ES256\"}",
    "dest":                                    "{\"kty\":\"EC\",\"crv\":\"P-
256\",\"x\":\"6MGDu3EsCdEJZVV2KFhnF2lxCRI5yNpf4vWQrCIMk5M\",\"y\":\"0OZbKAd
ooCqrQcPB3Bfqy0g-Y5SmnTyovFoFY35F00N\",\"alg\":\"ES256\"}",
    "encAlg": "A256GCM",
    "signingAlg": "ES256",
    "hashAlg": "SHA-256",
    "ledgerContractAddress": "0x7B7C7c0c8952d1BDB7E4D90B1B7b7C48c13355D1",
    "ledgerSignerAddress": "0x17bd12C2134AfC1f6E9302a532eFE30C19B9E903",
    "pooToPorDelay": 10000,
    "pooToPopDelay": 20000,
    "pooToSecretDelay": 150000
  },
  "signatures": {
     "providerSignature":
"eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9.S
XTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4g
WW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCw
gdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4.cu
22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2IpN6tPid6VrklHkq
```

```
sGqDqHCdP6O8TTB5dDDItllVo6_1pcbUrhiUSMxbbXUvdvWXzg-
UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRXe8P_ijQ7p8Vdz0TTrxUeT
3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQS
UmFAAjVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw",
    "consumerSignature":
"eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9.S
XTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4g
WW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCw
gdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4.cu
22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2IpN6tPid6VrklHkq
sGqDqHCdP6O8TTB5dDDItllVo6_1pcbUrhiUSMxbbXUvdvWXzg-
UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRXe8P_ijQ7p8Vdz0TTrxUeT
3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQS
UmFAAjVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw"
  }
}
```


Return type
raw_transaction

Example data
Content-Type: application/json


```
{
  "nonce": 46,
  "to": "0x4d722c3a1Cec53067106371034 95dDd9DFAda905",
  "from": "0xc6b8cf76bd7078e56c6ce8c357dd91caeea70170",
  "gasLimit": 12500000,
  "gasPrice": 1000,
  "chainId": 1337,
  "data":
"0x667a8beb0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000a000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000006737472696e6700000000000000000000000000000000000000000000000000
000000000"
}
```


Returns a raw transaction for the create agreement operation

```
POST /deploy_signed_transaction
```

Deploy signed transaction and send encrypted notification based on the event emitted by the DataSharingAgreement smart contract
Request body
body signed_transaction (required)

Example data
Content-Type: application/json


```
{
  "signedTransaction":
"0xf90f2a2e8203e883bebc20944d722c3a1cec5306710637103495ddd9dfada90580b90ec4
ee4b2db5000000000000000000000000000000000000000000000000000000000000003a0000
0000000000000000000000000000000000000000000000000079223a224543222c22637276223
a22502d323536222c2278223a22364d4744753345734364454a5a5656324b46686e46326c78
43524935794e706634765751724349 4d6b354d222c2279223a22304f5a624b41646f6f43717
251635042334266717930672d5935536d6e54796f76466f4659333546303 04e222c22616c67
223a224553323536227d0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000004036313333530646336666643730
326262393739333663383936386364694e415566385a6e7950456756466e3434325a644e716
9564a526d427172595258653850 5f5153556d4641416a564b5062427969377868 6f3075544f
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000673747
2696e6700000000000000000000000000000000000000000000000820a95a05263ad3d
490c6ab7baf8d755814ece3390de10e7df0cfc1ef3ae58361f949429a056fec9bcb23e8c1a1
cfd7d30f1c4959e63c1863ef1261b5941a9a22d779e855d"
}
```


Return type
transaction_object
Example data


```
{
  "transactionHash":
"0x833013a9428427016fc4b3cd1f05e9b42b289f4f98cd5bccfb91f4ae45fd630d",
  "transactionIndex": 0,
  "blockHash":
"0x1fd6a7de60041d0ec9c4735b9ecd8b022e8cbb154bc4f153cf9c517bc8f7e381",
  "blockNumber": 661175,
  "contractAddress": null,
  "cumulativeGasUsed": 1672030,
  "to": "0x4d722c3a1Cec5306710637103495dDd9DFAda905",
  "from": "0xC6b8cf76BD7078e56C6CE8C357dD91caeEa70170",
  "gasUsed": 1672030,
  "logsBloom":
"0x00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000002400000000000000000000000000000000000000000000000000800000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000001000000000000000000000000000000000000000000000008000000000
00000000000000080000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000",

"logs": [
    {
      "transactionIndex": 0,
      "blockNumber": 661175,
      "transactionHash":
"0x833013a9428427016fc4b3cd1f05e9b42b289f4f98cd5bccfb91f4ae45fd630d",
      "address": "0x4d722c3a1Cec5306710637103495dDd9DFAda905",
      "topics": [

"0x40f080228d46fb72660eddafe315e4a5b47df236dc33b76fcd122bcbea89b01d"
      ],
      "data":
"0x0000000000000000000000000000000000000000000000000000000000006000000000
```

```
0000000000000000000000000000000000000000000000000000000120000000000000000000
0000000000000000000000000000000000000000000000000000000f0000000000000000000000000000000000
0000000000000000000000000000000008c7b226b7479223a224543222c22637276223a22502
d323536222c2278223a22347378505070735a6f6d78506d5077444173715370393451705a33
695850387858345678574353436666d73222c2279223a223859495f627656724b50573633624
741734867527677584536756a33546c6e48776f5169395861454242455222c22616c67223a22
4553323536227d0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000008c7b226b7479223a224543222c22637276
223a22502d323536222c2278223a22364d474475334573436445 4a5a5656324b46686e46326
c7843524935794e7066347657517243494d6b354d222c2279223a22304f5a624b41646f6f43
7172516350423342667179306 72d5935536d6e54796f76466f4659333546303 04e222c22616
c67223a224553323536227d0000000000000000000000000000000000000000",
      "logIndex": 0,
      "blockHash":
"0x1fd6a7de60041d0ec9c4735b9ecd8b022e8cbb154bc4f153cf9c517bc8f7e381"
    }
  ],
  "confirmations": 1,
  "status": 1
}
```


Returns transaction receipt with confirmation 1

`GET /get_agreement/{agreement_id}`

Retrieve an agreement by agreement id
Path parameters
agreement_id (required)
Example data
Content-Type: application/json


```
  {
  "agreementId": 15,
  "providerPublicKey":                          "{\"kty\":\"EC\",\"crv\":\"P-
256\",\"x\":\"4sxPPpsZomxPmPwDAsqSp94QpZ3iXP8xX4VxWCSCfms\",\"y\":\"8YI_bvV
rKPW63bGAsHgRvwXE6uj3TlnHwoQi9XaEBBE\",\"alg\":\"ES256\"}",
  "consumerPublicKey":                          "{\"kty\":\"EC\",\"crv\":\"P-
256\",\"x\":\"6MGDu3EsCdEJZVV2KFhnF2lxCRI5yNpf4vWQrCIMk5M\",\"y\":\"0OZbKAd
ooCqrQcPB3Bfqy0g-Y5SmnTyovFoFY35F00N\",\"alg\":\"ES256\"}",
  "dataExchangeAgreementHash":
"61350dc3ffd702bb97936c8968d9fc19629a427157d6254bea5d415616edf07e",
  "dataOffering": {
    "dataOfferingId": "63662ebdb7d5dd78b7159566",
    "dataOfferingVersion": 0,
    "dataOfferingTitle": "Oil Supply Unit"
  },
  "purpose": "P&ID diagram of the Lube Oil supply Unit",
  "state": 0,
  "agreementDates": [
    1671753600,
    1786678869,
    1886678869
  ],
  "intendedUse": {
    "processData": true,
    "shareDataWithThirdParty": false,
    "editData": true
  },
```

```
  "licenseGrant": {
    "transferable": false,
    "exclusiveness": true,
    "paidUp": true,
    "revocable": true,
    "processing": true,
    "modifying": true,
    "analyzing": true,
    "storingData": true,
    "storingCopy": true,
    "reproducing": true,
    "distributing": false,
    "loaning": false,
    "selling": false,
    "renting": false,
    "furtherLicensing": false,
    "leasing": false
  },
  "dataStream": false,
  "personalData": false,
  "pricingModel": {
    "paymentType": "one-time purchase",
    "price": 125.68,
    "currency": "$",
    "fee": 6.28,
    "paymentOnSubscription": {
      "timeDuration": "string",
      "repeat": "string"
    },
    "isFree": false
  },
  "violation": {
    "violationType": 0
  },
  "signatures": {
    "providerSignature":
"eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9.S
XTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4g
WW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCw
gdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4.cu
22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2IpN6tPid6VrklHkq
sGqDqHCdP6O8TTB5dDDItllVo6_1pcbUrhiUSMxbbXUvdvWXzg-
UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRXe8P_ijQ7p8Vdz0TTrxUeT
3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQS
UmFAAjVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw",
    "consumerSignature":
"eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZXhhbXBsZSJ9.S
XTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IHlvdXIgZG9vci4g
WW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCw
gdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4.cu
22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2IpN6tPid6VrklHkq
sGqDqHCdP6O8TTB5dDDItllVo6_1pcbUrhiUSMxbbXUvdvWXzg-
UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRXe8P_ijQ7p8Vdz0TTrxUeT
3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQS
UmFAAjVKPbByi7xho0uTOcbH510a6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw"
  }
}
```

Returns the agreement by agreement id

```
GET /get_pricing_model/{agreement_id}
```

Retrieve an agreement's pricing model
pricingModel
Example data
Content-Type: application/json

```
  {
  "pricingModel": {
    "paymentType": "one-time purchase",
    "price": 125.68,
    "currency": "$",
    "fee": 6.28,
    "paymentOnSubscription": {
      "timeDuration": "string",
      "repeat": "string"
    },
    "isFree": false
  }
}
```

Returns the pricing model by agreement id

```
GET /check_active_agreements
```

Retrieve all the active agreements. (The agreements become active when they are created and stored on the blockchain.)
Returns a list of active agreements

```
GET /check_agreements_by_consumer/{consumer_public_keys}
/{active}
```

Retrieve all or just the active agreements of a consumer
Path parameters

- consumer_public_keys (required)
- active (required)

Example data

```
- [
      {"kty":"EC","crv":"P-
      256","x":"6MGDu3EsCdEJZVV2KFhnF2lxCRI5yNpf4vWQrCIMk5M","y":"0OZbKA
      dooCqrQcPB3Bfqy0g-Y5SmnTyovFoFY35F00M","alg":"ES256"}
    ]
- false
```

Return type
Returns all/active agreements based on consumer's public keys

```
GET /check_agreements_by_provider/{provider_public_keys}
/{active}
```

Retrieve all or just the active agreements of a provider

Path parameters

- provider_public_keys (required)
- active (required)

Example data

```
- [
    {"kty":"EC","crv":"P-
    256","x":"4sxPPpsZomxPmPwDAsqSp94QpZ3iXP8xX4VxWCSCfms","y":"8YI_bv
    VrKPW63bGAsHgRvwXE6uj3TlnHwoQi9XaEBBE","alg":"ES256"}
  ]
- true
```

Return type
Returns all/active agreements based on provider's public keys

```
GET /check_agreements_by_data_offering/{offering_id}
```

Retrieve all agreements for a data offering
Returns all agreements by offering id

```
GET /retrieve_agreements/{consumer_public_key}
```

Retrieve the active agreement by consumer public key whose start date is reached
Returns active agreement by consumer public key whose start date is reached

```
GET /state/{agreement_id}
```

Check the state of the agreement: active, violated, or terminated
Returns agreement's state based on agreement id

```
POST /evaluate_signed_resolution
```

Evaluate a signed resolution
body signed_resolution (required)

```
{
"proof":
"eyJhbGciOiJFUzI1NiJ9.eyJwcm9vZlR5cGUiOiJyZXNvbHV0aW9uIiwiZGF0YUV4Y2hhbmdlS
WQiOiJTTmg5eUtYYjJlaGxWSFJZQkllay16Z1pVaDJtU1NvMWpwbGg3SWEtNHlRIiwiaWF0Ijox
NjQ2OTUxNjM1LCJpc3MiOiJ7XCJhbGdcIjpcIkVTMjU2XCIsXCJjcnZcIjpcIlAtMjU2XCIsXCJ
kXCI6XCJ1Z1NpSTlJTEdnTWM1TmMwbkFhM3FGTjNBTjBvR2JhMzNJQWFrSHFkdm1nXCIsXCJrdH
lcIjpcIkVDXCIsXCJ4XCI6XCJMNldmVlhHYkgwaW82SnBtOTRTMWxwZGk2eUd0VDFPbVo2NUFfa
1NfaGs4XCIsXCJ5XCI6XCI2WUUwb1BPcFdCcUM3NURfanRKVWZ5NWxzWGxHak81ZzZRWGl2RHdN
REtjXCJ9Iiwic3ViIjoie1wiYWxnXCI6XCJFUzI1NlwiLFwiY3J2XCI6XCJQLTI1NlwiLFwia3R
5XCI6XCJFQ1wiLFwieFwiOlwiVlhzQnVPWndWamhvZkpWNGtBaGJhNnduMUVZRHdVSWtnWGIyZl
ZuTDh4Y1wiLFwieVwiOlwiaDRmTDVRdjRFWXQ3WGRLcWRJeTFaSnM0X1FXWURrWTF6VXpTb0k2M
U43WVwifSIsInJlc29sdXRpb24iOiJkZW5pZWQiLCJ0eXBlIjoiZGlzcHV0ZSJ9.TtxUm3E6Lfm
wEI74cr6RO4-nw-xcFaeARYOZ4z1dBVlc_JU0mCv0Ftr9tCDhggfLiJqb4RIPiNfIytFZMUbx-
g",
"sender_address": "0x4d82Bd33baA4Fe5489C45bBdC206019403dcF829"
}
```

Returns a raw transaction for the create agreement operation

```
POST /propose_penalty
```

Propose penalty
Request body
body choose_penalty (required)

```
{
  "agreementId": 15,
  "chosenPenalty": "NewEndDateForAgreementAndReductionOfPayment",
  "paymentPercentage": 16,
  "newEndDate": 189898999
}
```

Returns the chosen penalty and sends notification to the provider with the chosen penalty

```
PUT /enforce_penalty
```

Agree to penalty by enforcing it on the blockchain
Request body
body enforce_penalty (required)

```
{
  "senderAddress": "0xC6b8cf76BD7078e56C6CE8C357dD91caeEa70170",
  "agreementId": 15,
  "chosenPenalty": "NewEndDateForAgreementAndReductionOfPayment",
  "paymentPercentage": 16,
  "newEndDate": 189898999
}
```

Returns a raw transaction for the enforce penalty operation

```
PUT /terminate
```

Terminate agreement for batch data based on the last block of successful transfer and for streaming data if the end date is reached
body terminate (required)

```
{
  "senderAddress": "0xC6b8cf76BD7078e56C6CE8C357dD91caeEa70170",
  "agreementId": 15,
  "proof": "JWT",
}
```

Returns a raw transaction for the terminate agreement operation

Explicit consent:

```
POST /give_consent
```

Give consent to a user
body consent (required)

```
{
     "dataOfferingId": "63909dae0863a775a4d71bc9",
     "consentSubjects": [

      "did:ethr:i3m:0x026b23ab3cc76f1da1d5d2aa087d29894146ee52b56c23392a7f1
      36f7dc2a7a90c",

      "did:ethr:i3m:0x020bc2643908df0e6ab258a2dac38cd3b42ce2088a0a4e3b501d4
      85ababf9f5ad6",
     ],
     "consentFormHash":
   "36bede32098bd09e15a23274a37117e58a8b08bf54a1e48331a1ff8cc509e6da",
     "startDate": 1633344669,
     "endDate": 1673344669,
     "senderAddress": "0x9aDA42ff81B9D661cC4fdab62791DaC30cfe7305"
}
```

Returns a raw transaction for the give consent operation

```
PUT /revoke_consent
```

Revoke consent by consent subjects
body consent (required)

```
{
     "dataOfferingId": "63909dae0863a775a4d71bc9",
     "consentSubjects": [

      "did:ethr:i3m:0x026b23ab3cc76f1da1d5d2aa087d29894146ee52b56c23392a7f1
      36f7dc2a7a90c"
     ],
     "senderAddress": "0x9aDA42ff81B9D661cC4fdab62791DaC30cfe7305"
}
```

Returns a raw transaction for the enforce penalty operation

```
GET /check_consent_status/{dataOfferingId}
```

Retrieve consent status
Returns a list of consent status based on data offering and consent subject (optional)

```
POST /deploy_consent_signed_transaction
```

Deploy signed transaction and send encrypted notification based on the event emitted by the ExplicitUserConsent smart contract
body signed_transaction (required)
Returns transaction receipt with confirmation 1

7.6 Background Technologies

- **Hyperledger BESU:**

1	Technology
Technology name	**Hyperledger BESU**
Summary	Hyperledger BESU is an Ethereum client designed to be enterprise-friendly for both public and private permissioned network use cases. It can also be run on test networks such as Rinkeby, Ropsten, and Görli. Hyperledger BESU includes several consensus algorithms including PoW and PoA (IBFT, IBFT 2.0, Etherhash, and Clique). It also supports features including privacy and permissioning.
Description	Hyperledger BESU is an open-source Ethereum client developed under the Apache 2.0 license and written in Java. It runs on the Ethereum public networks, private networks, and test networks such as Rinkeby, Ropsten, and Görli. BESU implements Proof of Work (Ethash) and Proof of Authority (IBFT 2.0 and Clique) consensus mechanisms. BESU includes a command line interface and JSON-RPC API for running, maintaining, debugging, and monitoring nodes in an Ethereum network. BESU nodes support authentication and authorization, that is, identifying the user that performed the API query and allowing the execution of a specific set of methods. BESU supports two authentication mechanisms: username and password or JWT public key; see Figure 7.10. The communications are performed using the API via RPC over HTTP or via WebSockets. The API supports typical Ethereum functionalities such as: • ether mining; • smart contract development; • decentralized application (Dapp) development. The resultant BESU architecture is the following:

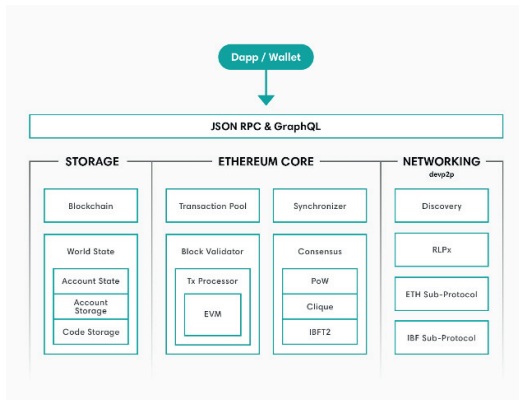
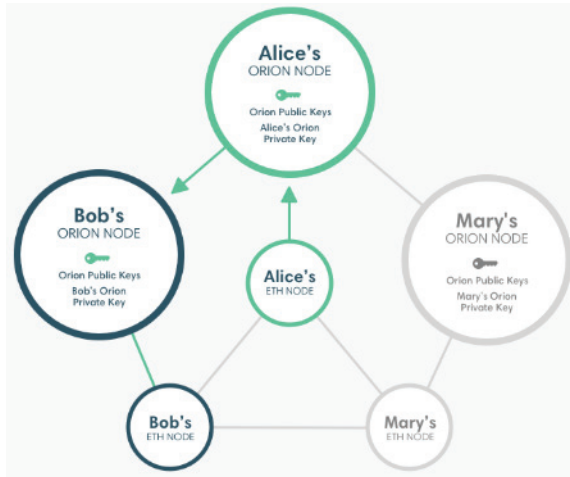


Figure 7.10 BESU architecture.

1	**Technology**
	BESU uses a private transaction manager, Orion, to implement privacy. Each BESU node sending or receiving private transactions requires an associated Orion node. Private transactions pass from the BESU node to the associated Orion node (see Figure 7.11). The Orion node encrypts and directly distributes (that is, point-to-point) the private transaction to the Orion nodes participating in the transaction.  **Figure 7.11** Alice sends a private transaction to Bob using Orion privacy manager. BESU also supports permissioning, which stands for permitting only specified nodes and accounts to participate by enabling node permissioning and account permissioning on the network. It supports local permissioning (a configuration file for each node) or on-chain (via smart contracts).
Keywords	Blockchain, distributed ledger, Ethereum, privacy, permissioning, authentication
ICT problem(s) and related functionality(ies)	Bullet list of the ICT problem(s) that the technology solves and associated functionalities. • **Distributed ledger** ○ Auditable data storage ○ Persistent transaction history ○ Permissioned and non-permissioned network ○ Pseudo-anonymous user identity • **Smart contracts** ○ Turing-complete machine ○ Immutable code (auditable and verifiable) • **Privacy** ○ Send cryptocurrency using private transactions ○ Execute smart contracts using private transactions

1	Technology
	• **Authentication** ○ JWT-based tokens ○ Username and password ○ JWT public key authentication • **Monitoring** ○ Visual representation of declining node or network performance ○ Collection of log files to enable issue diagnosis • **Communications** ○ Full-nodes and miners using HTTP/WebSockets ○ Encrypted communications for privacy (Orion) and signer (EthSigner) using TLS
TRL	Current technology readiness level of the technology: • TRL 7 – system prototype demonstration in operational environment
Website	https://www.hyperledger.org/projects/besu
Standards	BESU nodes are compatible with Ethereum public network. It supports different consensus protocols: Proof of Work (Ethash) and Proof of Authority (IBFT 2.0 and Clique). The communications use HTTP and JSON-RPC protocols. Clients can be authenticated using JWT. Smart contracts are coded using Solidity.

- **Solidity:**

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs that govern the behaviour of accounts within the Ethereum state. It is a curly-bracket language. It is designed to target the Ethereum virtual machine (EVM).

Solidity is used to develop the smart contracts that are deployed on the Ethereum blockchain.

- **Hardhat:**

Hardhat is a development environment for Ethereum software. It consists of different components for compiling, debugging, and deploying smart contracts, all of which work together to create a complete development environment.

Hardhat has a plug-in for integration with ethers.js, which is a compact library for interacting with the Ethereum blockchain.

- **Swagger:**

Swagger is a set of open-source rules and tools for developing RESTful APIs. It simplifies the process of writing APIs by specifying the standards and providing the tools required to write safe, performant, and scalable APIs. Moreover, the Swagger framework allows developers to create interactive, machine and human-readable API documentation.