Proofs Technical Design

General Information

As a User I want to be able to create Agreement, collect the signatures from the Signers. As a result I want Agreement to become Active.

Specification

🟦 **Glossary**

- *Creator* – creator of Agreement (i.e. the initiator of the Agreement signing process)
- *Signer* – user who is indicated in Agreement as a signer
- *Agreement File* – the original file of Agreement provided by *Creator*
- *Agreement File CID* – IPFS CID of the *Agreement File*
- *Agreement File Proof Data* – object with *Agreement CID* and other data for *Creator* to sign. For details see the structure of the file below
- *Agreement File Proof* – signed by *Creator*, *Agreement File Proof Data*. For details see the structure of the file below
- *Agreement File Proof* CID – IPFS CID of the signed *Agreement File Proof*
- *Agreement Sign Proof Data* – object with *Agreement CID* and other data for *Signer* to sign. For details see the structure of the file below
- *Agreement Sign Proof* – signed by any *Signer*, *Agreement Sign Proof Data* object. For details see the structure of the file below
- *Agreement Sign Proof CID* – IPFS CID of the signed *Agreement Sign Proof*
- *Agreement Proof* – object that contains *Agreement File Proof CID* along with *Agreement Sign Proof CIDs* of all *Signers*. For details see the

structure of the file below

- *Agreement Proof CID* – IPFS CID of the *Agreement Proof*

Note

For more details about the structure of *Agreement File Proof*, *Agreement File Proof Data*, *Agreement Sign Proof*, *Agreement Sign Proof Data*, and *Agreement Proof* – see a Medium article "EIP712 is here: What to expect and how to use it"

Agreement File Proof Data

```
{
  "types": {
    "EIP712Domain": [
        { "name": "name", "type": "string" },
        { "name": "version", "type": "string" },
        { "name": "chainId", "type": "uint64" },
        { "name": "verifyingContract", "type": "address" }
    ],
    "Agreement": [
      { "name": "from", "type": "address" },
      { "name": "agreementFileCID", "type": "string" },
      { "name": "signers", "type": "Signers" },
      { "name": "app", "type": "string" },
      { "name": "timestamp", "type": "uint64" },
      { "name": "metadata", "type": "string" }
    ],
    "Signers": [
      { "name": "address", "type": "string" },
      { "name": "metadata", "type": "string" }
    ]
  },
  "domain": {
    "name": "daosign",
    "version": "0.1.0"
  },
  "primaryType": "Agreement",
  "message": {
    "from": "<Creator's address>",
    "agreementFileCID": "<Agreement File CID>",
    "signers": [
```

```
      { "address": "<Signer 1 address>", "metadata": "{}" },
      { "address": "<Signer 2 address>", "metadata": "{}" },
      { "address": "<Signer 3 address>", "metadata": "{}" }
    ],
    "app": "daosign",
    "timestamp": <timestamp in seconds>,
    "metadata": "{}"
  }
}
```

Agreement File Proof

```
{
  "address": "<User's address>",
  "sig": "<User's signature of Agreement File Proof Data>",
  "data": <Agreement File Proof Data object>
}
```

Agreement Sign Proof Data

```
{
  "types": {
    "EIP712Domain": [
        { "name": "name", "type": "string" },
        { "name": "version", "type": "string" },
        { "name": "chainId", "type": "uint64" },
        { "name": "verifyingContract", "type": "address" }
    ],
    "Agreement": [
      { "name": "signer", "type": "address" },
      { "name": "agreementFileProofCID", "type": "string" },
      { "name": "app", "type": "string" },
      { "name": "timestamp", "type": "uint64" },
      { "name": "metadata", "type": "string" }
    ]
  },
  "domain": {
    "name": "daosign",
    "version": "0.1.0"
  },
  "primaryType": "Agreement",
  "message": {
    "signer": "<signer's address>",
```

```
    "agreementFileProofCID": "<Agreement File Proof CID>",
    "app": "daosign",
    "timestamp": <timestamp in seconds>,
    "metadata": "{}"
  }
}
```

> **Note**
>
> In the future we may want to extend the functionality of the platform to use Blockchain. In this case we may need to add `chainId` and `verifyingContract` fields in the `domain` block of *Agreement File Proof* and *Agreement Sign Proof*.
>
> Chain ID will enforce MetaMask to sign the payload only at the requested network, and Verifying Contract will indicate the address of a smart contract that can verify the signature.

```
    "chainId": 1,
    "verifyingContract": "0x0000000000000000000000000000000000000000"
```

Possible extension of Agreement File/Sign Proof Data

Agreement Sign Proof

```
{
  "address": "<signer's address>",
  "sig": "<signer's signature>",
  "data": <Agreement Sign Proof Data object>
}
```

Agreement Proof

```
{
  "agreementFileProofCID": "<Agreement File Proof CID>",
  "agreementSignProofs": [
      { "proofCID": "<Agreement Sign Proof CID>" },
      { "proofCID": "<Agreement Sign Proof CID>" },
      { "proofCID": "<Agreement Sign Proof CID>" }
  ],
  "timestamp": <timestamp in seconds; this this the last signature's timestamp
}
```

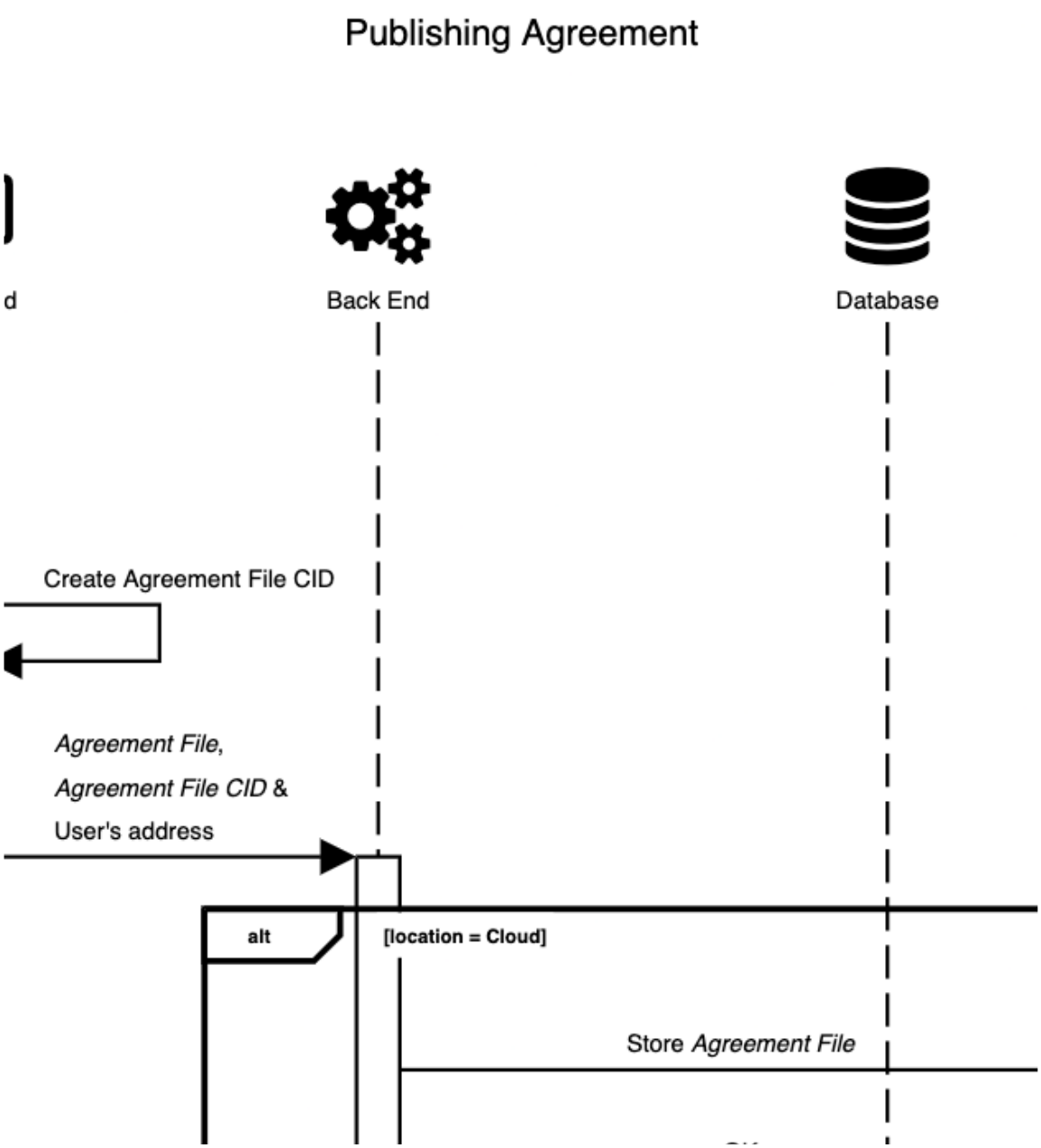
Description

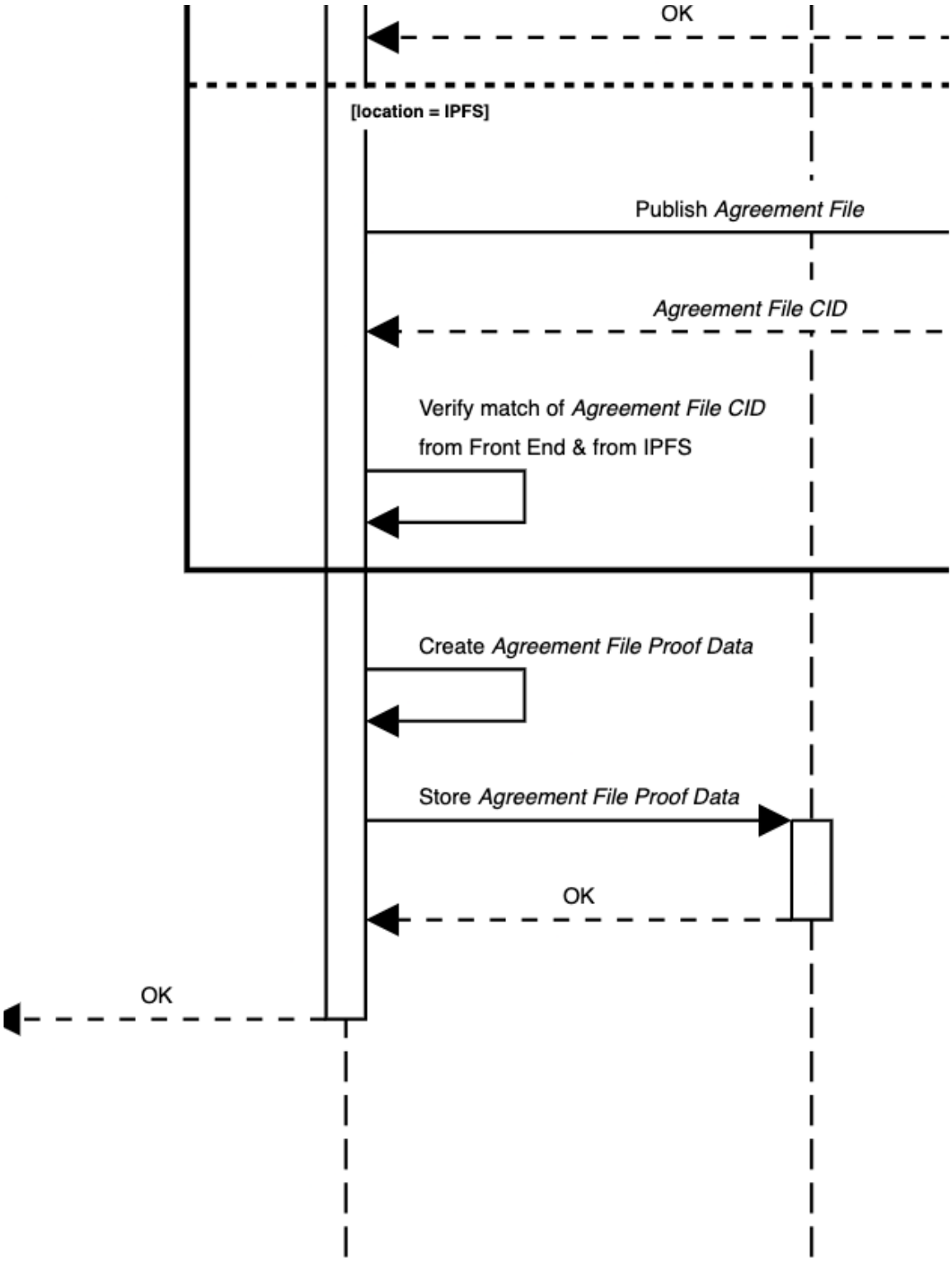
The Agreement signing process involves a couple of steps.

- Step 1 – Publishing Agreement
- Step 2 – Signing the document (a.k.a Proof-of-Authority)
- Step 3 – Signing Agreement by Signer(s) (a.k.a Proof-of-Signature)
- Step 4 – Create Agreement Proof (a.k.a Proof-of-Agreement)

Step 1 – Publishing Agreement

To create an Agreement, the user must first create an Agreement Draft. From the Agreement draft file *Agreement File CID* is created that is stored in the Database and IPFS. Also, the *Agreement File* (Agreement draft file) is stored in Amazon S3, if user want us to store the file.





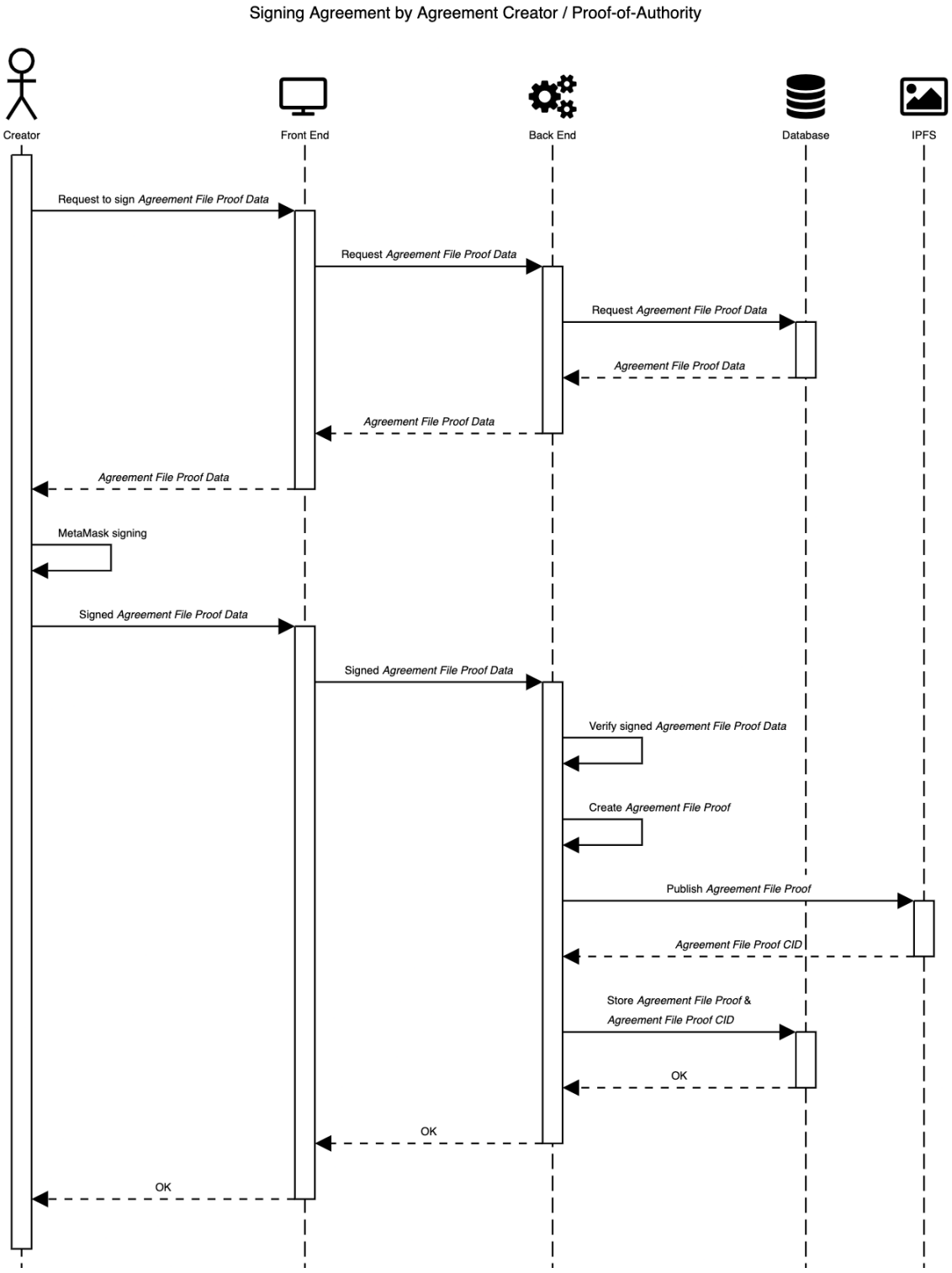
Step 2 – Signing the document (a.k.a Proof-of-Authority)

On this step, *Creator* of *Agreement File* has to proof that he's the one who created *Agreement File*.

Creator is presented with the *Agreement File Proof Data* to be signed. After *Creator* is satisfied with the document content, the user clicks the "Sign Agreement" button to sign the *Agreement File Proof Data*. Signed *Agreement File Proof Data* is stored in the Database and IPFS.

During *Agreement File Proof Data* signature verification step Back End should verify the following:

- timestamp in *Agreement File Proof Data* isn't older than 1 hour
- the object that the user has signed is the object that the user was asked to sign and it is unchanged
- the signature is valid i.e the user's address is the one who created the signature, and the signature is the signature of the *Agreement Sign File Data*

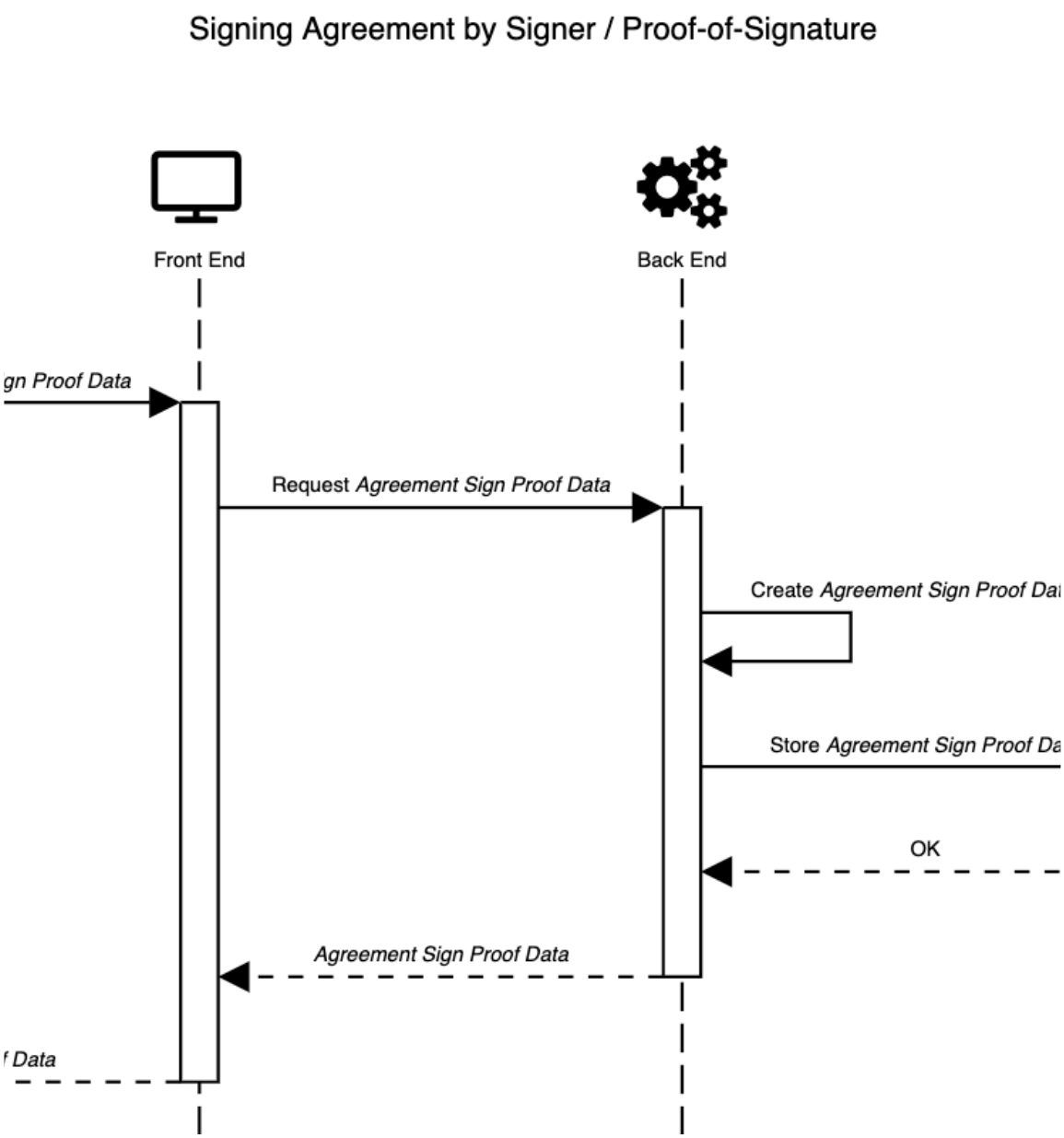


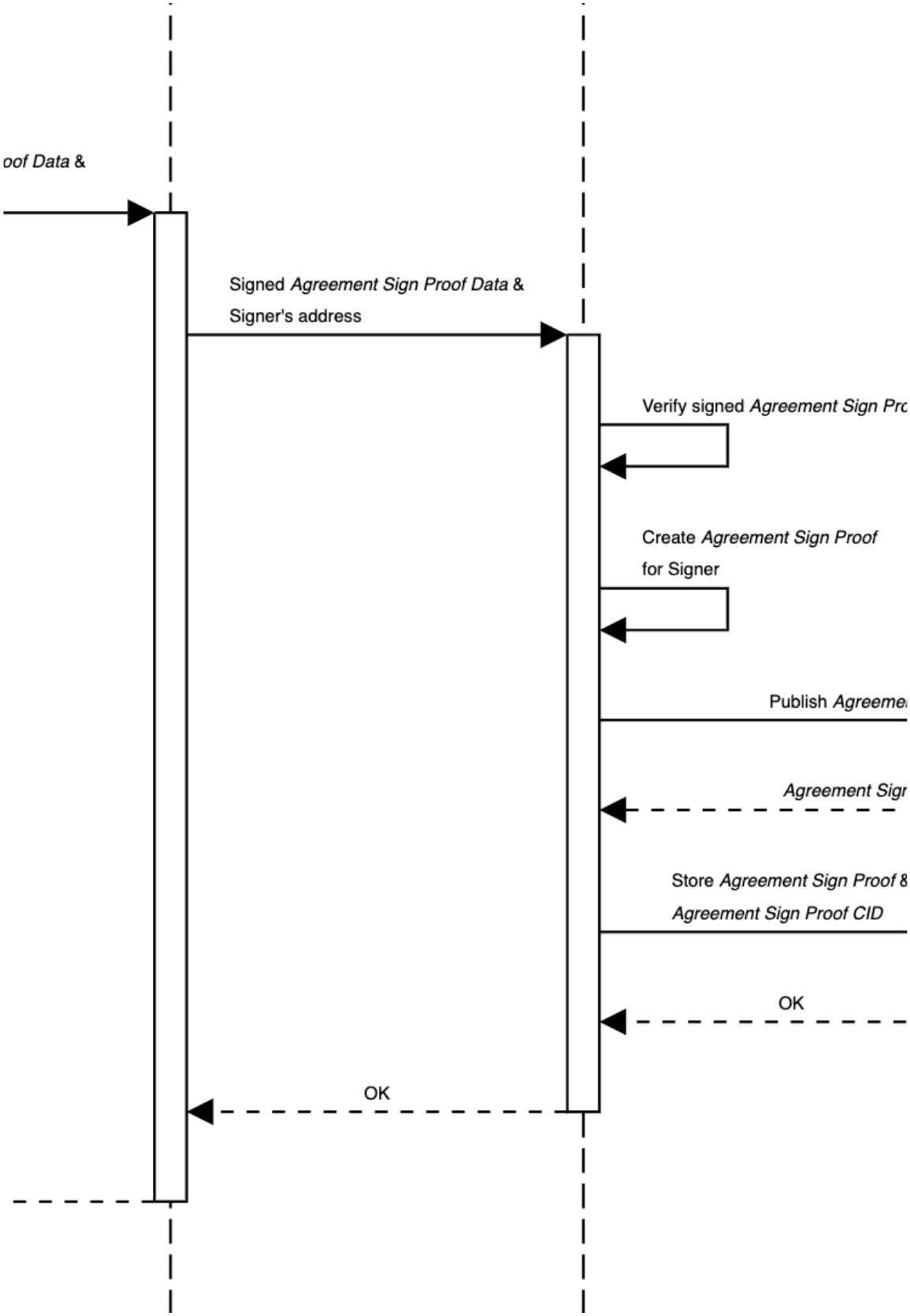
Step 3 – Signing Agreement by Signer(s) (a.k.a Proof-of-Signature)

Now, when *Agreement File Proof Data* is signed by *Creator*, the *Signers* of *Agreement* can start adding their signatures to approve the document. To do so, they will receive *Agreement Sign Proof Data* to sign. Their signature proofs (*Agreement Sign Proofs*) will be stored to Database and IPFS.

During *Agreement Sign Proof Data* signature verification step Back End should verify the following:

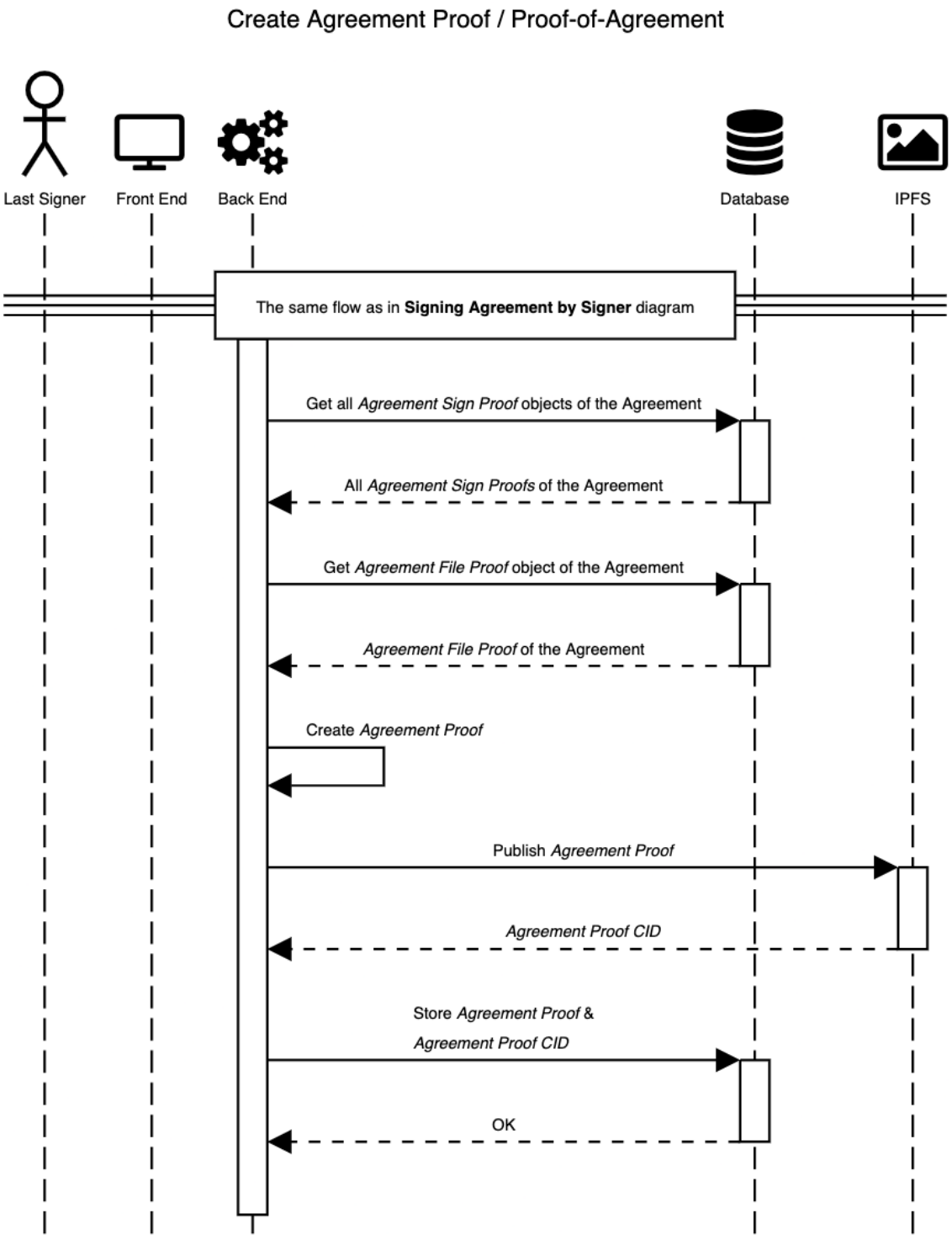
- timestamp in *Agreement Sign Proof Data* isn't older than 1 hour
- the object that the user has signed is the object that the user was asked to sign and it is unchanged
- the signature is valid i.e the user's address is the one who created the signature, and the signature is the signature of the *Agreement Sign Proof Data*





Step 4 – Create Agreement Proof (a.k.a Proof-of-Agreement)

On this final step, *Agreement File Proof* signed by *Creator* is combined with *Agreement Sign Proofs* from all *Signers*. Based on these proofs the final proof, *Agreement Proof*, is created. It is also stored to IPFS and the Database. After this, the Agreement is considered to be fully signed and becomes active.



Diagrams Source Files

To open the source files please use SequenceDiagram.org

4 Create Agree… oof.txt
18 Jan 2023, 01:25 PM

3 Signing Agree…ner.txt
18 Jan 2023, 01:25 PM

2 Signing Agree…tor.txt
18 Jan 2023, 01:25 PM

1 Publish
18 Jan 20

+ Add label

Be the first to add a reaction