Ockr Specification

Bormann, Fabian
fabian.bormann@ockr.io

November 26, 2023

Contents

1 Abstract

Ockr aims to set a standard for creating machine-readable and reliable documents, enabling the verification of their authenticity. In short Ockr tries to tackle three problems:

1. How can we make sure that a document is machine-readable?
2. How can we ensure that a document has not been modified?
3. How can we ensure that a document really has been issued by a certain authority?

2 Motivation

Most of the documents today are created in a digital form and are shared online. The file itself serves as a vehicle for the actual information. File hashes are often used to prove the authenticity of the information. However, the file hash can be changed over time, while the information remains the same. Compression techniques are used behind the scenes while sharing the files online, as well as storing the file in cloud environments. Finally printing the file and scanning it again will result in a different file hash. Images or documents that store the information in images are not machine readable. While process automation became crutual in many industries, many processes still rely on manual work because of the lack of machine readable documents. OCR (Optical Character Recognition) is a technique that can be used to extract the information from images, but it is not reliable enough to be used in critical processes as it can not guarantee the authenticity of the information. Ockr aims to solve these problems by providing a standard for creating machine-readable and reliable information.

3 Process

To decouple the information from the file, the raw information needs to be provided and a hash of the information needs to be calculated. This hash will be used to verify the authenticity of the information and stored within a QR code on the document. Besides the hash, the QR code will also contain metadata helping a machine to re-create the information.

3.1 Document creation

As most of the documents are created digitally, the raw text can be extracted from the file such as DOCX, XLSX, PDF and other formats. In case there is no raw text available, OCR can be used to extract the text from the document but a human would need to verify the text.

3.1.1 Information extraction

To ensure a good user experience, plugins for common document editors will be provided to extract the raw information from the document and to create the Ockr QR code. The plugins will be open source and can be used as a reference implementation for other plugins.

3.2 OCR verification

There is no internet connection needed and no additional service required to verify the ORC result. Therefore data pipelines can ensure to have a reliable OCR result in critical processes. An algorithm breaks down the document in different parts and verifies the OCR result of each part. The algorithm can be implemented in any programming language and can be used in any environment. Besides the main hash of the information a hash of each part will be calculated and stored within the QR code. There are chars that are often misinterpreted by OCR like 0 and O. With having the hash of each part, different combinations of chars can be tried to find the correct result. Algorithms and OCR models will be provided as open source projects under the ockr.io umbrella.

3.3 Guarantee authenticity

An open source backend service will be publicly available to register an information hash on a blockchain. The service will be available as a REST API. Once the informantion hash has been settled on-chain, it can be verified by anyone. The service will be open source and can be used as a reference implementation for other services.

3.4 Authorship verification

The authority that registered the information would also need a registration event on-chain. Anyone can register a new authority but to give a certain authority more credibility, the authority can be verified by other authorities.

3.5 Trust Trees

The trust tree is a graph of authorities that trust each other. An algorithm will be provided to calculate the trust tree and its score. The score can be used to determine the credibility of an authority. The REST API will also offer an endpoint to check if a certain authority is part of the trust tree of another authority. If for example, an automated HR process wants to verify a certificate, it can check if the university that issued the certificate is trusted by the government.

4 Algorithms

Algorithms need to ensure that an OCR result is reliable. The first set of algorithms will be used to extract the information from the document. Besides of the OCR algorithm itself, they will also include algorithms to build the information hash from the OCR output. Those will be open source and are part of the Ockr umbrella. They provive something like the syntax for the OCR scan results. The second set of algorithms will be used to give the results a semantic. Those are called templates and they are optional. Templates will not be part of any repository. They can be registered on-chain using the REST API. Templates can be used to give an automated process the relevant values. For example a value relative to certain anchor words could defined as the product weight on a delivery note to calculate the shipping costs for an automated process.

4.1 OCR Algorithms

PP-OCRv4-mobile and PP-OCRv4-server are part of PaddleOCR and will be provided as ONNX models in the Ockr model zoo. The model zoo will be an open source repository where everyone can contribute OCR models or enhance the existing ones. A model should run isolated and wrapped by a REST API capsulated in a Docker container. The blueprint will use Fast API and the Docker image will be provided alongside to the code as a reference implementation.

4.2 Hash Algorithms

SHA256 will be used as the hash algorithm. The hash will be calculated from the intial information during the content creation process. In the verification processs it will be calculated from the OCR result and a naive string concatenation approach. As this solution is not reliable enough, SHA256 will be applied also to the output of a post-processing method, introduced as puzzle algorithms.

4.3 Puzzle Algorithms

The puzzle algorithms will be used to break down the image in different parts. Each part will be hashed by SHA256 and stored within the QR code. The puzzle algorithms will be open source and can be used as a reference implementation for other algorithms. A simple and default algorithm will be described in the following section.

4.3.1 Default Puzzle Algorithm

The default algorithm takes the image and splits it into n parts. It takes x and y as parameters to define the number of parts. The image will be split into x parts horizontally and y parts vertically. The parts will be hashed by SHA256 and stored within the QR code.

Algorithm 1 Default Puzzle Algorithm

```
Require: $OcrResults$
Require: $x \geq 1$
Require: $y \geq 1$
  $subHashes \leftarrow []$
  for $i \leftarrow x$ to $x$ do
      for $j \leftarrow y$ to $y$ do
          $results \leftarrow getOcrResultsUnderArea(x, y)$
          $hash \leftarrow SHA256(concat(results))$
          $subHashes \leftarrow append(hash)$
      end for
  end for
  return $subHashes$
```

4.4 Templates

A template consists of a set of anchors, such as the OR code or other required words, and a set of keys including releative positions to a subset of anchors. The anchors are used to find the values for the keys in the OCR result. The values can be defined as a match to a key, or as a regex matching the key. The template structure will be defined as a JSON schema and can be registered on-chain using the REST API and a Cardano matadata standard. An offline template editor will be provided to create and edit templates. The editor will be open source and can be used as a reference implementation for other validators or editors.

```
interface Template {
    anchors: Anchor[];
    keys: Key[];
}

interface Anchor {
    type: AnchorType;
    reference: BoderSide | string;
}

enum BorderSide {
    Top,
    Right,
    Bottom,
    Left
}

enum AnchorType {
    QRCode,
    Text,
    Border
}

// regex is optional otherwise the full ocr result will be used
interface Key {
    name: string;
    regex: string;
    matcher: {
        anchor: number;
        position: Position;
    }
}

// relative to page width between -1 and 1
interface Position {
    x: number;
    y: number;
    tolerance: number;
}
```

5 Implementation

There are different parts mentioned in the algorithm and process section. This section will describe the implementation of those parts and the concret implementation.

5.1 Ockr REST API

The Ockr REST API is the main interface to interact with the Ockr ecosystem. It is used to register information hashes on-chain, to register authorities and to register templates. The API is implemented using Java Sprint Boot. It will be available as an open source project under the Ockr umbrella. The API will be available as a Docker image which allows companies to run it in their own environment instead of using the public API. The public API will be available at *api.ockr.io/v1*.

5.1.1 Endpoints

```
// Registers a new information hash on-chain

POST /register/hash

// Request Body
{
    "hash": "string",
    "signature": "string",
    "template": "string" | undefined,
    "subHashes": ["string"] | undefined,
    "algorithm": "string" | undefined,
}

// Registers a new template on-chain
// Template is a JSON schema described in the algorithms section
// The template will be minified and validated

POST /register/template

// Request Body
{
    "template": Template,
    "signature": "string",
}

// Registers a new authority on-chain
// The authority will be co-signed by the registering service as the
    first validator

POST /register/authority
```

```
// Request Body
{
    "name": "string",
    "signature": "string",
}

// Returns the trust tree of an authority if the hash matches with a
    registered information hash on-chain

POST /verify/hash

// Request Body
{
    "hash": "string",
    "template": "string" | undefined,
    "subHashes": ["string"] | undefined,
}

// Returns all templates that have been registered on-chain

GET /template

// Returns a template by id

GET /template/{id}

// Returns all models connected to the api

GET /model

// Connects a model to the api

POST /connect/model
```