Digitally signed degree certificates

# Introduction

The system provides a web portal where a student can download his/her degree and grade-card in PDF format. Each document is digitally signed by the Registrar and then the Director.

It uses Python Flask based backend that provides API services to download degree and grade-card and an HTML Javascript based web portal.

# Backend

The backend API service is a Flask based Python application. The student data that contains each student’s name, roll number, last 4 digits of Aadhar number and the marks in 5 subjects are stored as a CSV file. In future, this data can be taken from a DB but for the project’s simplicity, it is in a CSV.

The API call GET /get\_document is called from the UI with the following parameters:

* name: The name of the student
* roll\_num: The roll number of the student
* aadhar\_last\_4\_digits: The last 4 digits of the Aadhar number of the student
* doc\_type: The document type, which can be either ‘Degree’ or ‘Grade-card’

The 1st 3 of these values are matched with the records in the CSV file. If any record is found with all 3 matching values, the record is accessed. Else the API returns an error.

The record along with doc\_type is passed to the method create\_pdf(). This method uses an html string, that depends on the doc\_type and provides a template for the Degree or the Grade-Card. The name, roll number and marks in 5 subjects are populated in this template. Then the Python library ‘weasyprint’ is used to generate a PDF document from the html template.

Once the PDF is created, we pass it to the method create\_watermarked\_pdf() to add the institute’s watermark to the PDF. This method uses PdfFileWriter & PdfFileReader from the package ‘PyPDF4’ and canvas from the package ‘reportlab.pdfgen’. Using canvas, we create a temporary image of the watermark in memory that is read into a PdfFileReader object. The original PDF is read into another PdfFileReader object and then a merge call is made to merge both the objects. The result is written into a PDF using PdfFileWriter to create the watermarked PDF which is returned.

The generated PDF file is passed to the method generate\_signature() to generate the digital signature of the Registrar. In this method, the PDF file is read and the read bytes are passed to the method sha256() in the package ‘hashlib’. This returns the SHA256 hash.

Then a GET API call is made to https://worldtimeapi.org/api/timezone/Etc/UTC to get the current UTC time. Worldtimeapi is a well known and reputed provider of time data. The fact that the API request uses https ensures that the current time is retrieved securely. The timestamp returned from this API is concatenated to the SHA256 hash. The concatenated string along with the Registrar’s private key is passed to the method encrypt() in our implementation RSA\_cryptosystem.py. This method does an RSA encryption of the string using the Registrar’s private key. This encrypted value is the Registrar’s digital signature.

Then the PDF file is passed to the method generate\_signature(), but this time along with the Registrar’s signature. The PDF file is read. Then the Registrar’s signature is encoded using encode('utf-8') and concatenated to the bytes of the read PDF file. Then the Director’s signature is generated for the resultant concatenated bytes of the PDF file+Registrar’s signature, by first generating the hash, concatenating the current time-stamp to the hash and then doing RSA encryption using the Director’s private key to get the Director’s signature. This ensures that the Director’s signature is generated only AFTER the Registrar has signed.

The PDF file is then returned as an attachment in response to the API call. The two signatures and PDF filename are passed as custom headers in the API response.

An after\_request(response) is added to the API implementation to expose the custom headers in 'Access-Control-Expose-Headers'.

The API is deployed at http://127.0.0.1:5001/get\_document

# Frontend

The frontend is a HTML Javascript implementation. Three inputs are collected from the user, namely name, roll\_num and aadhar\_last\_4\_digits. Then the user is provided two buttons, ‘Download Degree’ and ‘Download Grade-Card’. Clicking either of these buttons calls the function downloadPDF() with a value ‘1’ for Degree and ‘2’ for Grade-card. The downloadPDF() is defined as an async function, to allow usage of await. Depending on the parameter value, await fetch() is called on either of the following URLs:

<http://127.0.0.1:5001/get_document?name=$>{name}&roll\_num=${roll\_num}&aadhar\_last\_4\_digits=${aadhar\_last\_4\_digits}&doc\_type=Degree

<http://127.0.0.1:5001/get_document?name=$>{name}&roll\_num=${roll\_num}&aadhar\_last\_4\_digits=${aadhar\_last\_4\_digits}&doc\_type=Grade-Card

Once the result of the await fetch() is received, signature1, signature2 and filename are read from the response headers. Then the PDF file sent as an attachment in the API response is downloaded. The two signatures are displayed on screen in appropriate output-boxes.

# Verification of signatures

A python application, verify\_signature.py is used to verify the digital signatures. It collects three inputs as follows:

* File path of the downloaded file
* Registrar’s digital signature
* Director’s digital signature

The user can cut and paste the signatures from the UI to the verification app.

A Public Key Authority pkda.py is run as a separate Python Flask API. This PKDA can provide the Registrar and Director’s public keys securely after encrypting them with the PKDA’s private key.

The verify\_signature app requests the PKDA for the Registrar and Director’s public keys with the following API calls:

GET http://127.0.0.1:5002/get\_public\_key?client\_id=registrar

GET http://127.0.0.1:5002/get\_public\_key?client\_id=director

The verify\_signature app has the PKDA’s public key and uses it to decrypt the response of the above API calls to get the Registrar and Director’s public keys.

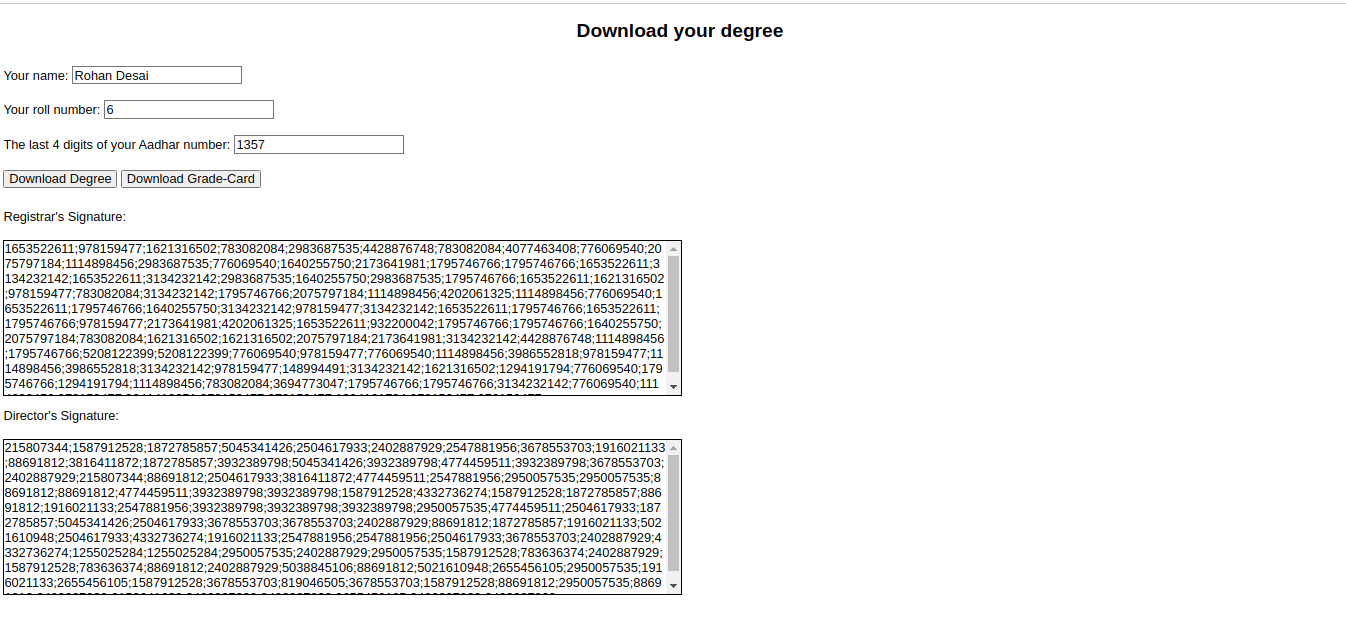
Using the Registrar’s public key, the app decrypts the Registrar’s signature. The decrypted string contains two parts, the PDF’s hash and the timestamp of signature, separated by ‘||’. The two are split and displayed on screen.

Then the PDF file is read and its SHA256 hash is generated. This is then matched with the PDF’s hash that was extracted in the above step. If the two match, then the Registrar signature is valid.

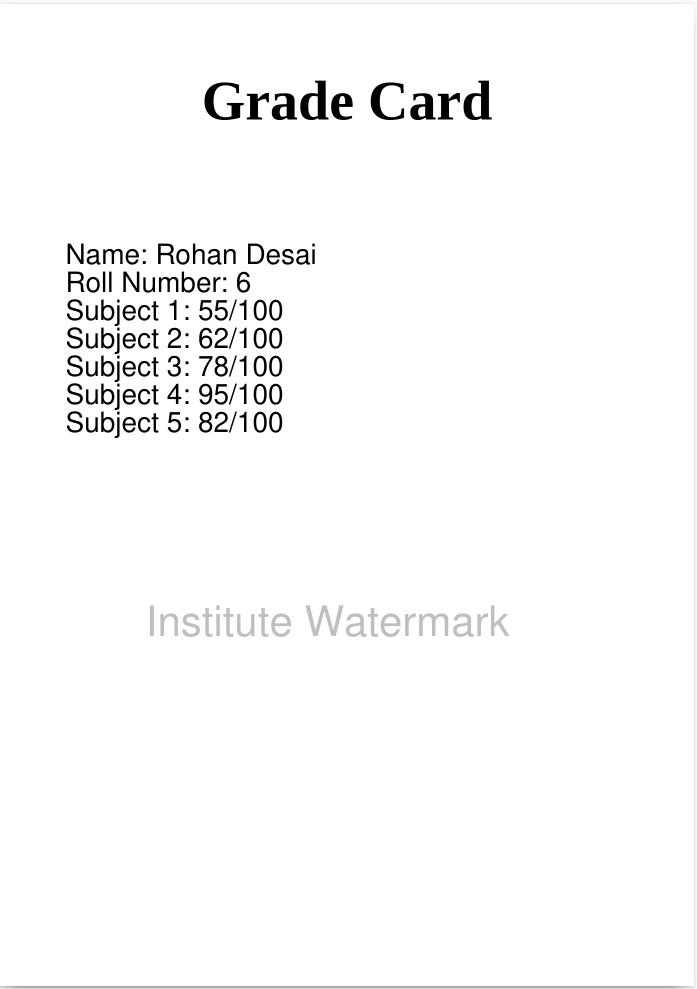
Then using the Director’s public key, the app decrypts the Director’s signature. The decrypted string contains two parts, the hash and the timestamp of signature, separated by ‘||’. The two are split and displayed on screen.

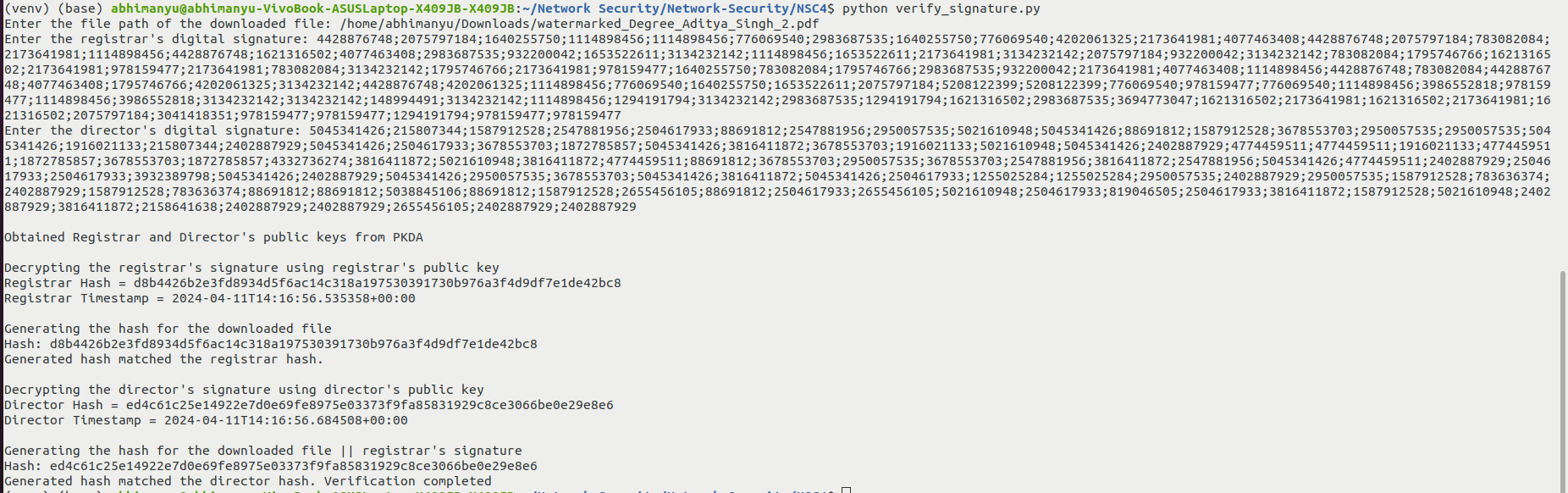
Then the PDF file is read and the Registrar’s signature is concatenated to it. The SHA256 hash of the resultant bytes is generated. If this matches the hash generated above, then the Director’s signature is valid and proved to be generated after the signature of the Registrar (since the hash was of the PDF + the Registrar’s signature).

# Screenshots

The above is a screenshot of the UI once the download is complete.

The above is an image of the degree that is downloaded.

The above is an image of a Grade-Card.

The above is an image of the verify\_signature app’s output.