

INTERN PROJECT REPORT
ON
Digital Signature Using Python

Submitted by:

Janmejay Mohanty

Under the Guidance of

MR. SHANTANU CHATTERJEE

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN
Computer Science & Engineering



Faculty of Engineering & Technology

**MANAV RACHNA INTERNATIONAL INSTITUTE OF
RESEARCH AND STUDIES, Faridabad
NAAC ACCREDITED 'A' GRADE**

JUNE, 2020

Declaration

I (We) hereby declare that this project report entitled “**Digital Signature Using Python**” by **Janmejay Mohanty**, being submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science and Engineering** under Faculty of Engineering & Technology of Manav Rachna International Institute of Research and Studies, Faridabad, during the academic year June,2019, is a bonafide record of our original work carried out under guidance and supervision of **Mr. Shantanu Chatterjee** and has not been presented elsewhere.

Janmejay Mohanty

TABLE OF CONTENTS

Declaration	i
Abstract	ii
Chapter	
I. Introduction	
II. Problem Definition and Requirement Analysis	
2.1 Problem Statement	
2.2 Approach For Resolving Problem Statement	
2.3 Requirement Analysis	
III. Design and Implementation	
3.1 Introduction	
3.2 Flow Chart	
3.3 Source Code	
3.4 Directories	
IV. Project Output	
V. Conclusion and Future Enhancements	
5.1 Summary of work done	
5.2 Proposal/scope of future enhancement	
References/Bibliography	

ABSTRACT

This is an overview of my project on which I had done work. My project title is Digital Signature Using Python using ECC (Elliptical Curve Cryptography). My project use python code as well as its environment. I had used ECDSA (Elliptical Curve Digital Signature Algorithm) library which is present in python libraries.

Basically, in my project we take any document file and get it digitally signed with Digital Signature which is generated by ECDSA algorithm which makes the document authentic digitally.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA), where it is a digital signature scheme designed to provide a digital signature based on a secret key which is known only to the signer and also on the actual message being signed. Digital signature schemes reduce transmission costs, because the message is contained in the signature itself and no separate message and signature need be sent again. In this project report, we try to provide more secure digital signature scheme by using python based elliptic curve cryptography (ECC). In particular, we introduce ECDSA library.

Introduction

Our objective is to digitally signing any file containing text or data with private and public keys of Elliptical Curve Cryptography. Elliptic-curve cryptography is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC allows smaller keys compared to non-EC cryptography to provide equivalent security.

We have used ECDSA (Elliptic Curve Digital Signature Algorithm) library which is available in python and used it for research purpose on digital signature using ECC (Elliptical Curve Cryptography). In cryptography, the Elliptic Curve Digital Signature Algorithm offers a variant of the Digital Signature Algorithm which uses elliptic curve cryptography.

The whole coding of digital signature using ECC is done on python platform and also it uses some additional modules such as Django, MongoDB, Html and CSS templates for making an interactive interface for our project of digital signature using ECC.

Python is an interpreted, high-level, general-purpose programming language. Django is a Python-based free and open-source web framework that follows the model-template-view architectural pattern. MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices.

Problem Definition and Requirement Analysis

2.1 Problem Statement

A digital signature scheme allows one to sign an electronic message and later the produced signature can be validated by the owner of the message or by any verifier. Most of the existing digital signature schemes were developed based on a single hard problem like factoring, discrete logarithm, residuality or elliptic curve discrete logarithm problems. Although these schemes appear secure, one day in a near future they may be exploded if one finds a solution of the single hard problem.

2.2 Approach for Resolving Problem Statement

To overcome this problem, in this study, we proposed a new signature scheme based on multiple hard problems namely factoring and discrete logarithms. We combined both signing and verifying equations from ECDSA library from python library such that it generates two secret keys (public and private keys) which is depend on each other.

2.3 Requirement Analysis

Software

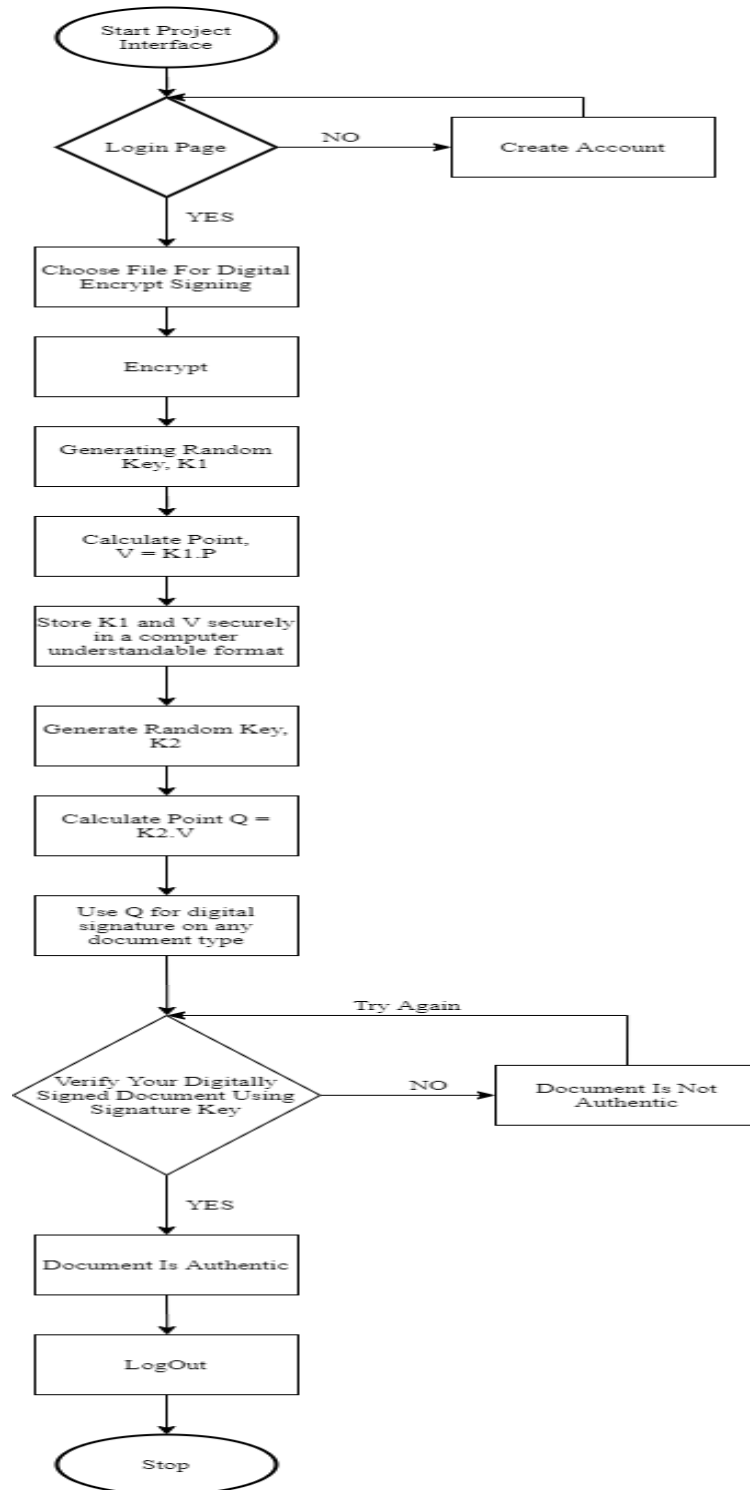
1. Database Required (i.e. MongoDB).
2. ECDSA (Elliptic Curve Digital Signature Algorithm) Python Library.
3. Python Runtime Environment.
4. Django Framework.
5. Operating systems: Windows 7 or later, macOS, and Linux

Hardware

1. Internet Connection Required.
2. Processors: Intel Core i3 processor or later.
3. Disk space: 1 GB.

Design And Implementation

3.1 Flow chart



First the virtual server is created by Django.

Login Page: Login screen is loaded for the user so that they can access the project. If login fails then user can create a new account. If login successful then user can proceed further.

Now user can choose any file they want to digital sign. After they choose file, they can select encrypt option for signing the file and for cross verification they can choose verify option. A pop-up message is shown on the screen from which user can see whether their file is securely digital signed or not.

Explanation of ECC (Elliptical Curve Cryptography)

The ECDSA (Elliptic Curve Digital Signature Algorithm) is a cryptographically secure digital signature scheme, based on the elliptic-curve cryptography (ECC). ECDSA relies on the math of the cyclic groups of elliptic curves over finite fields and on the difficulty of the ECDLP problem (elliptic-curve discrete logarithm problem). The ECDSA sign / verify algorithm relies on EC point multiplication and works as described below. ECDSA keys and signatures are shorter than in RSA for the same security level. A 256-bit ECDSA signature has the same security strength like 3072-bit RSA signature.

ECDSA uses cryptographic elliptic curves (EC) over finite fields in the classical Weierstrass form. These curves are described by their EC domain parameters, specified by various cryptographic standards such as SECG: SEC 2 and Brainpool (RFC 5639). Elliptic curves, used in cryptography, define:

Generator point G , used for scalar multiplication on the curve (multiply integer by EC point)

Order n of the subgroup of EC points, generated by G , which defines the length of the private keys (e.g. 256 bits)

For example, the 256-bit elliptic curve secp256k1 has:

Order $n =$

1157920892373161954235709850086879078528375642790749043826051631
41518161494337 (prime number)

Generator point $G \{x =$

5506626302227734366957871889516853432625060345377759417550018736
0389116729240, $y =$

3267051002075881697808308513050704318447127338065924327593890433
5757337482424}

Key Generation

The ECDSA key-pair consists of:

private key (integer): `privKey`

public key (EC point): `pubKey = privKey * G`

The private key is generated as a random integer in the range $[0..n-1]$. The public key `pubKey` is a point on the elliptic curve, calculated by the EC point multiplication: `pubKey = privKey * G` (the private key, multiplied by the generator point `G`).

The public key EC point $\{x, y\}$ can be compressed to just one of the coordinates + 1 bit (parity). For the `secp256k1` curve, the private key is 256-bit integer (32 bytes) and the compressed public key is 257-bit integer (~ 33 bytes).

ECDSA Sign

The ECDSA signing algorithm (RFC 6979) takes as input a message `msg` + a private key `privKey` and produces as output a signature, which consists of pair of integers $\{r, s\}$. The ECDSA signing algorithm is based on the ElGamal signature scheme and works as follows (with minor simplifications):

1. Calculate the message hash, using a cryptographic hash function like SHA-256: $h = \text{hash}(\text{msg})$
2. Generate securely a random number k in the range $[1..n-1]$
3. In case of deterministic-ECDSA, the value k is HMAC-derived from $h + \text{privKey}$
4. Calculate the random point $R = k * G$ and take its x-coordinate: $r = R.x$
5. Calculate the signature proof: $s = \kappa^{-1} * (h + r * \text{privKey})(\text{mod } n)$
6. The modular inverse $k^{-1}(\text{mod } n)$ is an integer, such that $k * k^{-1} \equiv (\text{mod } n)$
7. Return the signature $\{r, s\}$.

The calculated signature $\{r, s\}$ is a pair of integers, each in the range $[1..n-1]$. It encodes the random point $R = k * G$, along with a proof s , confirming that the signer knows the message h and the private key `privKey`. The proof s is by idea verifiable using the corresponding `pubKey`.

ECDSA signatures are 2 times longer than the signer's private key for the curve used during the signing process. For example, for 256-bit elliptic curves (like secp256k1) the ECDSA signature is 512 bits (64 bytes) and for 521-bit curves (like secp521r1) the signature is 1042 bits.

ECDSA Verify Signature

The algorithm to verify a ECDSA signature takes as input the signed message msg + the signature {r, s} produced from the signing algorithm + the public key pubKey, corresponding to the signer's private key. The output is boolean value: valid or invalid signature. The ECDSA signature verify algorithm works as follows (with minor simplifications):

1. Calculate the message hash, with the same cryptographic hash function used during the signing: $h = \text{hash}(\text{msg})$
2. Calculate the modular inverse of the signature proof: $s1 = s^{-1}(\text{mod } n)$
3. Recover the random point used during the signing: $R' = (h * s1) * G + (r * s1) * \text{pubKey}$
4. Take from R' its x-coordinate: $r' = R'.x$
5. Calculate the signature validation result by comparing whether $r' == r$

The general idea of the signature verification is to recover the point R' using the public key and check whether it is same point R, generated randomly during the signing process.

Working Procedure of ECDSA

The ECDSA signature {r, s} has the following simple explanation:

The signing signing encodes a random point R (represented by its x-coordinate only) through elliptic-curve transformations using the private key privKey and the message hash h into a number s, which is the proof that the message signer knows the private key privKey. The signature {r, s} cannot reveal the private key due to the difficulty of the ECDLP problem.

The signature verification decodes the proof number s from the signature back to its original point R, using the public key pubKey and the message hash h and compares the x-coordinate of the recovered R with the r value from the signature.

The Mathematical logic behind the ECDSA Sign / Verify

The equation behind the recovering of the point R', calculated during the signature verification, can be transformed by replacing the pubKey with privKey * G as follows:

$$\begin{aligned} R' &= (h * s1) * G + (r * s1) * \text{pubKey} = \\ &= (h * s1) * G + (r * s1) * \text{privKey} * G = \\ &= (h + r * \text{privKey}) * s1 * G \end{aligned}$$

If we take the number $s = k^{-1} * (h + r * \text{privKey})(\text{mod } n)$, calculated during the signing process, we can calculate $s1 = s^{-1}(\text{mod } n)$ like this:

$$\begin{aligned} s1 &= s^{-1}(\text{mod } n) = \\ &= (k^{-1} * (h + r * \text{privKey}))^{-1}(\text{mod } n) = \\ &= k * (h + r * \text{privKey})^{-1}(\text{mod } n) \end{aligned}$$

Now, replace s1 in the point R'.

$$\begin{aligned} R' &= (h + r * \text{privKey}) * s1 * G = \\ &= (h + r * \text{privKey}) * k * (h + r * \text{privKey})^{-1}(\text{mod } n) * G = \\ &= k * G \end{aligned}$$

The final step is to compare the point R' (decoded by the pubKey) with the point R (encoded by the privKey). The algorithm in fact compares only the x-coordinates of R' and R: the integers r' and r.

It is expected that $r' == r$ if the signature is valid and $r' \neq r$ if the signature or the message or the public key is incorrect.

ECDSA: Public Key Recovery from Signature

It is important to know that the ECDSA signature scheme allows the public key to be recovered from the signed message together with the signature. The recovery process is based on some mathematical computations (described in the SECG: SEC 1 standard) and returns 0, 1 or 2 possible EC points that are valid public keys, corresponding to the signature. To avoid this ambiguity, some ECDSA implementations add one additional bit v to the signature during the

signing process and it takes the form $\{r, s, v\}$. From this extended ECDSA signature $\{r, s, v\}$ + the signed message, the signer's public key can be restored with confidence.

The public key recovery from the ECDSA signature is very useful in bandwidth constrained or storage constrained environments (such as blockchain systems), when transmission or storage of the public keys cannot be afforded. For example, the Ethereum blockchain uses extended signatures $\{r, s, v\}$ for the signed transactions on the chain to save storage and bandwidth.

Public key recovery is possible for signatures, based on the ElGamal signature scheme (such as DSA and ECDSA).

3.3 Source Code

ECDSA.py

```
from ecdsa import SigningKey, NIST521p

sk = SigningKey.generate(curve=NIST521p)

vk = sk.verifying_key

with open("private.pem", "wb") as f:

    f.write(sk.to_pem())

with open("public.pem", "wb") as f:

    f.write(vk.to_pem())

with open("private.pem") as f:

    sk = SigningKey.from_pem(f.read())

with open("message.txt", "rb") as f:

    message = f.read()

sig = sk.sign(message)

with open("signature", "wb") as f:

    f.write(sig)
```

verify.py

```
from ecdsa import VerifyingKey, BadSignatureError

vk = VerifyingKey.from_pem(open("public.pem").read())

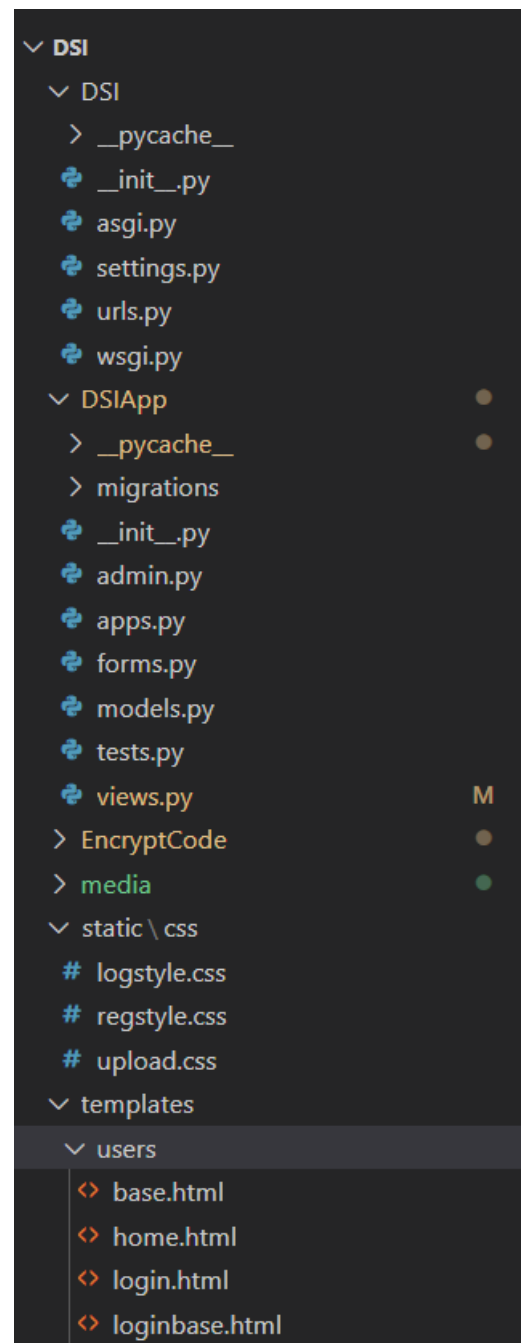
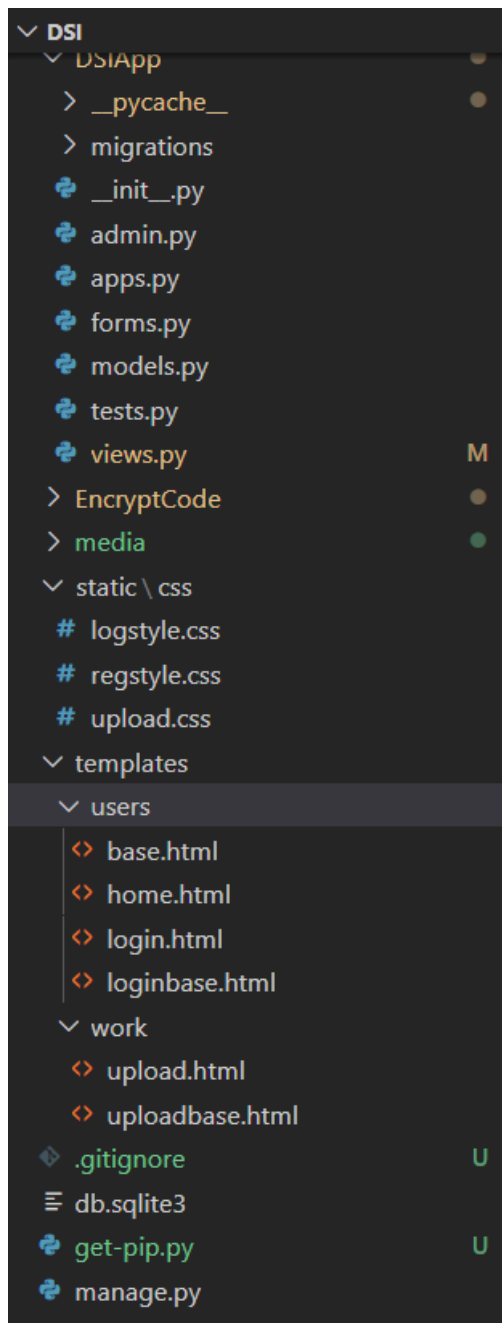
with open("message.txt", "rb") as f:
    message = f.read()

with open("signature", "rb") as f:
    sig = f.read()

try:
    vk.verify(sig, message)

    print ("good signature")
except BadSignatureError:
    print ("BAD SIGNATURE")
```

3.4 Directories



Code of Interface and Configuration files of Digital Signature Using Python

In DSI Directory

settings.py

```
"""
Django settings for DSI project.

Generated by 'django-admin startproject' using Django 3.0.6.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')
STATIC_DIR = os.path.join(BASE_DIR, 'static')

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '#hgtw7ss*gp2#@_3*1qql_z60h2l1n3_2yzh9_u4-^odg1-*36'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'DSIApp',
    'widget_tweaks',
```



```

    'rest_framework_mongoengine',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'DSI.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR,],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]

WSGI_APPLICATION = 'DSI.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
#     }
# }

DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'DSIDATA',
        'CLIENT': {
            'host': 'mongodb+srv://dsiuser:drdo@cluster0-

```

```
7hfqh.mongodb.net/DSIDATA?retryWrites=true&w=majority',
    'username': 'dsiuser',
    'password': 'drdo',
    'authSource': 'admin',
  }
}
}
```

```
# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-password-validators
```

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/
```

```
LANGUAGE_CODE = 'en-us'
```

```
TIME_ZONE = 'UTC'
```

```
USE_I18N = True
```

```
USE_L10N = True
```

```
USE_TZ = True
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/
```

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    STATIC_DIR,
]
```

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'
```

urls.py

```
"""DSI URL Configuration
```

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/3.0/topics/http/urls/>

Examples:

Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path("", views.home, name='home')

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path("", Home.as_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

```
"""
```

```
from django.contrib import admin
from django.urls import path
from DSIApp import views
from django.views.generic.base import RedirectView
from django.conf import settings
from django.conf.urls.static import static
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', views.log_view, name = 'login'),
    path('submit/', views.form_view),
    path('upload/', views.upload_view, name = 'upload'),
    path("", RedirectView.as_view(url='login')),
]
```

```
if settings.DEBUG:
```

```
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

In DSIApp Directory

admin.py

```
from django.contrib import admin
from DSIApp.models import Student , log
# Register your models here.
class StudentAdmin(admin.ModelAdmin):
    list_display = ['Name','Age','Mob','Add','Email','Pass','Re_Pass']
admin.site.register(Student,StudentAdmin)
```

apps.py

```
from django.apps import AppConfig
```

```
class DsiappConfig(AppConfig):
    name = 'DSIApp'
```

forms.py

```
from django import forms
from DSIApp.models import Student , log
```

```
class LogForm(forms.ModelForm):
    class Meta:
        model = log
        widgets = {'Password': forms.PasswordInput()}
        fields = '__all__'
```

```
class StudentForm(forms.ModelForm):
    class Meta:
        model = Student
        widgets = {'Pass': forms.PasswordInput(),'Re_Pass': forms.PasswordInput()}
        fields = '__all__'
```

models.py

```
from django.db import models
```

```
class log(models.Model):
    Username = models.CharField(max_length=25)
    Password = models.CharField(max_length=25)
```

```

class Student(models.Model):
    Name = models.CharField(max_length=25)
    Age = models.IntegerField()
    Email = models.CharField(max_length=25, null=True)
    Mob = models.IntegerField()
    Add = models.CharField(max_length=64)
    Pass = models.CharField(max_length=25, null=True)
    Re_Pass = models.CharField(max_length=25, null=True)

```

views.py

```

from django.shortcuts import render , redirect
from DSIApp.models import Student
from DSIApp.forms import StudentForm, LogForm
from django.contrib import messages
from pymongo import MongoClient
from ecdsa import SigningKey, NIST521p, VerifyingKey, BadSignatureError
from django.core.files.storage import FileSystemStorage
# Create your views here.

```

```

n = "

```

```

def log_view(request):

```

```

    if request.method == 'POST':
        log_form = LogForm(request.POST)
        myclient = MongoClient("mongodb+srv://dsiuser:drdo@cluster0-7hfqh.mongodb.net/DSIDATA?retryWrites=true&w=majority")
        username1 = request.POST['Username']
        password1 = request.POST['Password']
        mydb = myclient["DSIDATA"]
        mycol = mydb["DSIApp_student"]
        myquery = { "Name": username1,"Pass":password1 }
        mydoc = mycol.find(myquery)

        for x in mydoc:
            global n
            n = x['Name']
            return redirect("/upload")
    else:
        log_form = LogForm()
    return render(request,'users/loginbase.html',{ 'log_form': log_form })

```

```

def form_view(request):

```

```

    if request.method == 'POST':
        form = StudentForm(request.POST)
        if form.is_valid():
            password1 = request.POST['Pass']
            password2 = request.POST['Re_Pass']

```

```

        if password1 != password2:
            return redirect('/')
        else:
            form.save()
            return redirect('login')
    else:
        form = StudentForm()
    return render(request, 'users/home.html', {'form': form})

def upload_view(request):
    context = {'a': n}
    if request.method == "POST" and 'sbutton' in request.POST:
        uploaded_file = request.FILES['document']
        fs = FileSystemStorage()
        name = fs.save(uploaded_file.name, uploaded_file)
        context['url'] = fs.url(name)

    if request.method == "POST" and 'ebutton' in request.POST:
        uploaded_file = request.FILES['document']
        sk = SigningKey.generate(curve=NIST521p)
        vk = sk.verifying_key
        with open("EncryptCode/private.pem", "wb") as f:
            f.write(sk.to_pem())
        with open("EncryptCode/public.pem", "wb") as f:
            f.write(vk.to_pem())

        with open("EncryptCode/private.pem") as f:
            sk = SigningKey.from_pem(f.read())
            message = uploaded_file.read()
            sig = sk.sign(message)
            with open("EncryptCode/signature", "wb") as f:
                f.write(sig)

    if request.method == "POST" and 'vbutton' in request.POST:
        uploaded_file = request.FILES['document']
        message = uploaded_file.read()
        vk = VerifyingKey.from_pem(open("EncryptCode/public.pem").read())
        with open("EncryptCode/signature", "rb") as f:
            sig = f.read()
        try:
            vk.verify(sig, message)
            print ("good signature")
            messages.success(request, "Good Signature")
        except BadSignatureError:
            print ("BAD SIGNATURE")
            messages.success(request, "Bad Signature")

    if request.method == "POST" and 'lbutton' in request.POST:
        return redirect("login")
    return render(request, 'work/upload.html', context)

```

In static\css Directory

logstyle.css

```
.login-page {
    width: 360px;
    padding: 8% 0 0;
    margin: auto;
}
.form {
    position: relative;
    z-index: 1;
    background: #FFFFFF;
    max-width: 360px;
    margin: 0 auto 100px;
    padding: 45px;
    text-align: center;
    box-shadow: 0 0 20px 0 rgba(0, 0, 0, 0.2), 0 5px 5px 0 rgba(0, 0, 0, 0.24);
}

.form input {
    font-family: "Roboto", sans-serif;
    outline: 0;
    background: #f2f2f2;
    width: 100%;
    border: 0;
    margin: 0 0 15px;
    padding: 15px;
    box-sizing: border-box;
    font-size: 14px;
}

.form button {
    font-family: "Roboto", sans-serif;
    text-transform: uppercase;
    outline: 0;
    background: #4CAF50;
    width: 100%;
    border: 0;
    padding: 15px;
    color: #FFFFFF;
    font-size: 14px;
    -webkit-transition: all 0.3 ease;
    transition: all 0.3 ease;
    cursor: pointer;
}

.form button:hover, .form button:active, .form button:focus {
    background: #43A047;
}

.message {
```

```

    margin: 15px 0 0;
    color: #b3b3b3;
    font-size: 12px;
}
.message a {
    color: #4CAF50;
    text-decoration: none;
}

body {
    background: #76b852; /* fallback for old browsers */
    background: -webkit-linear-gradient(right, #76b852, #8DC26F);
    background: -moz-linear-gradient(right, #76b852, #8DC26F);
    background: -o-linear-gradient(right, #76b852, #8DC26F);
    background: linear-gradient(to left, #76b852, #8DC26F);
    font-family: "Roboto", sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}

```

regstyle.css

```

/*
=====
=====
#FONT
=====
===== */
.font-robo {
    font-family: "Roboto", "Arial", "Helvetica Neue", sans-serif;
}

.font-poppins {
    font-family: "Poppins", "Arial", "Helvetica Neue", sans-serif;
}

/*
=====
=====
#GRID
=====
===== */
.row {
    display: -webkit-box;
    display: -webkit-flex;

```



```
display: -moz-box;
display: -ms-flexbox;
display: flex;
-webkit-flex-wrap: wrap;
-ms-flex-wrap: wrap;
flex-wrap: wrap;
}
```

```
.row-space {
  -webkit-box-pack: justify;
  -webkit-justify-content: space-between;
  -moz-box-pack: justify;
  -ms-flex-pack: justify;
  justify-content: space-between;
}
```

```
.col-2 {
  width: -webkit-calc((100% - 30px) / 2);
  width: -moz-calc((100% - 30px) / 2);
  width: calc((100% - 30px) / 2);
}
```

```
@media (max-width: 767px) {
  .col-2 {
    width: 100%;
  }
}
```

```
/*
```

```
=====
=====
```

#BOX-SIZING

```
=====
===== */
```

```
/**
```

```
* More sensible default box-sizing:
* css-tricks.com/inheriting-box-sizing-probably-slightly-better-best-practice
*/
```

```
html {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
```

```
* {
  padding: 0;
  margin: 0;
}
```

```
*, *:before, *:after {
  -webkit-box-sizing: inherit;
  -moz-box-sizing: inherit;
  box-sizing: inherit;
}
```

```
/*
```

```
=====
=====
```

```
  #RESET
```

```
=====
===== */
```

```
/**
```

```
 * A very simple reset that sits on top of Normalize.css.
```

```
 */
```

```
body,
h1, h2, h3, h4, h5, h6,
blockquote, p, pre,
dl, dd, ol, ul,
figure,
hr,
fieldset, legend {
  margin: 0;
  padding: 0;
}
```

```
/**
```

```
 * Remove trailing margins from nested lists.
```

```
 */
```

```
li > ol,
li > ul {
  margin-bottom: 0;
}
```

```
/**
```

```
 * Remove default table spacing.
```

```
 */
```

```
table {
  border-collapse: collapse;
  border-spacing: 0;
}
```

```
/**
```

```
 * 1. Reset Chrome and Firefox behaviour which sets a `min-width: min-content;`
```

```
 *   on fieldsets.
```

```
 */
```

```
fieldset {
  min-width: 0;
  /* [1] */
```

```
border: 0;
}
```

```
button {
  outline: none;
  background: none;
  border: none;
}
```

```
/*
```

```
=====
```

```
=====
```

```
#PAGE WRAPPER
```

```
=====
```

```
===== */
```

```
.page-wrapper {
  min-height: 100vh;
}
```

```
body {
  font-family: "Poppins", "Arial", "Helvetica Neue", sans-serif;
  font-weight: 400;
  font-size: 14px;
}
```

```
h1, h2, h3, h4, h5, h6 {
  font-weight: 400;
}
```

```
h1 {
  font-size: 36px;
}
```

```
h2 {
  font-size: 30px;
}
```

```
h3 {
  font-size: 24px;
}
```

```
h4 {
  font-size: 18px;
}
```

```
h5 {
  font-size: 15px;
}
```

```
h6 {
  font-size: 13px;
}
```

```
/*
```

```
=====
```

```
=====
```

```
#BACKGROUND
```

```
=====
```

```
===== */
```

```
.bg-blue {
  background: #2c6ed5;
}
```

```
.bg-red {
  background: #fa4251;
}
```

```
.bg-gra-01 {
  background: -webkit-gradient(linear, left bottom, left top, from(#fbc2eb), to(#a18cd1));
  background: -webkit-linear-gradient(bottom, #fbc2eb 0%, #a18cd1 100%);
  background: -moz-linear-gradient(bottom, #fbc2eb 0%, #a18cd1 100%);
  background: -o-linear-gradient(bottom, #fbc2eb 0%, #a18cd1 100%);
  background: linear-gradient(to top, #fbc2eb 0%, #a18cd1 100%);
}
```

```
.bg-gra-02 {
  background: -webkit-gradient(linear, left bottom, right top, from(#fc2c77), to(#6c4079));
  background: -webkit-linear-gradient(bottom left, #fc2c77 0%, #6c4079 100%);
  background: -moz-linear-gradient(bottom left, #fc2c77 0%, #6c4079 100%);
  background: -o-linear-gradient(bottom left, #fc2c77 0%, #6c4079 100%);
  background: linear-gradient(to top right, #fc2c77 0%, #6c4079 100%);
}
```

```
/*
```

```
=====
```

```
=====
```

```
#SPACING
```

```
=====
```

```
===== */
```

```
.p-t-100 {
  padding-top: 100px;
}
```

```
.p-t-130 {
  padding-top: 130px;
}
```

```
.p-t-180 {
  padding-top: 180px;
}
```

```
.p-t-20 {
  padding-top: 20px;
}
```

```
.p-t-15 {
  padding-top: 15px;
}
```

```
.p-t-10 {
  padding-top: 10px;
}
```

```
.p-t-30 {
  padding-top: 30px;
}
```

```
.p-b-100 {
  padding-bottom: 100px;
}
```

```
.m-r-45 {
  margin-right: 45px;
}
```

```
/*
```

```
=====
=====
```

```
#WRAPPER
```

```
=====
===== */
```

```
.wrapper {
  margin: 0 auto;
}
```

```
.wrapper--w960 {
  max-width: 960px;
}
```

```
.wrapper--w780 {
  max-width: 780px;
}
```

```
.wrapper--w680 {
  max-width: 680px;
}
```

/*

=====

#BUTTON

===== */

```
.btn {  
  display: inline-block;  
  line-height: 50px;  
  padding: 0 50px;  
  -webkit-transition: all 0.4s ease;  
  -o-transition: all 0.4s ease;  
  -moz-transition: all 0.4s ease;  
  transition: all 0.4s ease;  
  cursor: pointer;  
  font-size: 18px;  
  color: #fff;  
  font-family: "Poppins", "Arial", "Helvetica Neue", sans-serif;  
}
```

```
.btn--radius {  
  -webkit-border-radius: 3px;  
  -moz-border-radius: 3px;  
  border-radius: 3px;  
}
```

```
.btn--radius-2 {  
  -webkit-border-radius: 5px;  
  -moz-border-radius: 5px;  
  border-radius: 5px;  
}
```

```
.btn--pill {  
  -webkit-border-radius: 20px;  
  -moz-border-radius: 20px;  
  border-radius: 20px;  
}
```

```
.btn--green {  
  background: #57b846;  
}
```

```
.btn--green:hover {  
  background: #4dae3c;  
}
```

```
.btn--blue {  
  background: #4272d7;
```

```
}
```

```
.btn--blue:hover {  
  background: #3868cd;  
}
```

```
/*
```

```
/*
```

```
=====
```

```
=====
```

```
#FORM
```

```
=====
```

```
===== */
```

```
input {  
  outline: none;  
  margin: 0;  
  border: none;  
  -webkit-box-shadow: none;  
  -moz-box-shadow: none;  
  box-shadow: none;  
  width: 100%;  
  font-size: 14px;  
  font-family: inherit;  
}
```

```
.input--style-4 {  
  line-height: 50px;  
  background: #fafafa;  
  -webkit-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);  
  -moz-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);  
  box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);  
  -webkit-border-radius: 5px;  
  -moz-border-radius: 5px;  
  border-radius: 5px;  
  padding: 0 20px;  
  font-size: 16px;  
  color: #666;  
  -webkit-transition: all 0.4s ease;  
  -o-transition: all 0.4s ease;  
  -moz-transition: all 0.4s ease;  
  transition: all 0.4s ease;  
}
```

```
.input--style-4::-webkit-input-placeholder {  
  /* WebKit, Blink, Edge */  
  color: #666;  
}
```

```
.input--style-4:-moz-placeholder {  
  /* Mozilla Firefox 4 to 18 */  
  color: #666;  
  opacity: 1;  
}
```

```
.input--style-4::-moz-placeholder {  
  /* Mozilla Firefox 19+ */  
  color: #666;  
  opacity: 1;  
}
```

```
.input--style-4:-ms-input-placeholder {  
  /* Internet Explorer 10-11 */  
  color: #666;  
}
```

```
.input--style-4:-ms-input-placeholder {  
  /* Microsoft Edge */  
  color: #666;  
}
```

```
.label {  
  font-size: 16px;  
  color: #555;  
  text-transform: capitalize;  
  display: block;  
  margin-bottom: 5px;  
}
```

```
.radio-container {  
  display: inline-block;  
  position: relative;  
  padding-left: 30px;  
  cursor: pointer;  
  font-size: 16px;  
  color: #666;  
  -webkit-user-select: none;  
  -moz-user-select: none;  
  -ms-user-select: none;  
  user-select: none;  
}
```

```
.radio-container input {  
  position: absolute;  
  opacity: 0;  
  cursor: pointer;  
}
```

```
.radio-container input:checked ~ .checkmark {
```



```

    background-color: #e5e5e5;
}

.radio-container input:checked ~ .checkmark:after {
    display: block;
}

.radio-container .checkmark:after {
    top: 50%;
    left: 50%;
    -webkit-transform: translate(-50%, -50%);
    -moz-transform: translate(-50%, -50%);
    -ms-transform: translate(-50%, -50%);
    -o-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
    width: 12px;
    height: 12px;
    -webkit-border-radius: 50%;
    -moz-border-radius: 50%;
    border-radius: 50%;
    background: #57b846;
}

.checkmark {
    position: absolute;
    top: 50%;
    -webkit-transform: translateY(-50%);
    -moz-transform: translateY(-50%);
    -ms-transform: translateY(-50%);
    -o-transform: translateY(-50%);
    transform: translateY(-50%);
    left: 0;
    height: 20px;
    width: 20px;
    background-color: #e5e5e5;
    -webkit-border-radius: 50%;
    -moz-border-radius: 50%;
    border-radius: 50%;
    -webkit-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
    -moz-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
    box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
}

.checkmark:after {
    content: "";
    position: absolute;
    display: none;
}

.input-group {

```

```
position: relative;
margin-bottom: 22px;
}
```

```
.input-group-icon {
position: relative;
}
```

```
.input-icon {
position: absolute;
font-size: 18px;
color: #999;
right: 18px;
top: 50%;
-webkit-transform: translateY(-50%);
-moz-transform: translateY(-50%);
-ms-transform: translateY(-50%);
-o-transform: translateY(-50%);
transform: translateY(-50%);
cursor: pointer;
}
```

```
/*
```

```
=====
=====
```

```
#SELECT2
```

```
=====
===== */
```

```
.select--no-search .select2-search {
display: none !important;
}
```

```
.rs-select2 .select2-container {
width: 100% !important;
outline: none;
background: #fafafa;
-webkit-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
-moz-box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
box-shadow: inset 0px 1px 3px 0px rgba(0, 0, 0, 0.08);
-webkit-border-radius: 5px;
-moz-border-radius: 5px;
border-radius: 5px;
}
```

```
.rs-select2 .select2-container .select2-selection--single {
outline: none;
border: none;
height: 50px;
background: transparent;
```

```

}

.rs-select2 .select2-container .select2-selection--single .select2-selection__rendered {
  line-height: 50px;
  padding-left: 0;
  color: #555;
  font-size: 16px;
  font-family: inherit;
  padding-left: 22px;
  padding-right: 50px;
}

.rs-select2 .select2-container .select2-selection--single .select2-selection__arrow {
  height: 50px;
  right: 20px;
  display: -webkit-box;
  display: -webkit-flex;
  display: -moz-box;
  display: -ms-flexbox;
  display: flex;
  -webkit-box-pack: center;
  -webkit-justify-content: center;
  -moz-box-pack: center;
  -ms-flex-pack: center;
  justify-content: center;
  -webkit-box-align: center;
  -webkit-align-items: center;
  -moz-box-align: center;
  -ms-flex-align: center;
  align-items: center;
}

.rs-select2 .select2-container .select2-selection--single .select2-selection__arrow b {
  display: none;
}

.rs-select2 .select2-container .select2-selection--single .select2-selection__arrow:after {
  font-family: "Material-Design-Iconic-Font";
  content: '\f2f9';
  font-size: 24px;
  color: #999;
  -webkit-transition: all 0.4s ease;
  -o-transition: all 0.4s ease;
  -moz-transition: all 0.4s ease;
  transition: all 0.4s ease;
}

.rs-select2 .select2-container.select2-container--open .select2-selection--single .select2-
selection__arrow::after {
  -webkit-transform: rotate(-180deg);
}

```

```
-moz-transform: rotate(-180deg);
-ms-transform: rotate(-180deg);
-o-transform: rotate(-180deg);
transform: rotate(-180deg);
}
```

```
.select2-container--open .select2-dropdown--below {
  border: none;
  -webkit-border-radius: 3px;
  -moz-border-radius: 3px;
  border-radius: 3px;
  -webkit-box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);
  -moz-box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);
  box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);
  border: 1px solid #e0e0e0;
  margin-top: 5px;
  overflow: hidden;
}
```

```
.select2-container--default .select2-results__option {
  padding-left: 22px;
}
```

```
/*
```

```
=====
=====
#TITLE
```

```
=====
===== */
.title {
  font-size: 24px;
  color: #525252;
  font-weight: 400;
  margin-bottom: 40px;
}
```

```
/*
```

```
=====
=====
#CARD
```

```
=====
===== */
.card {
  -webkit-border-radius: 3px;
  -moz-border-radius: 3px;
  border-radius: 3px;
  background: #fff;
}
```

```
.card-4 {  
  background: #fff;  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  border-radius: 10px;  
  -webkit-box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);  
  -moz-box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);  
  box-shadow: 0px 8px 20px 0px rgba(0, 0, 0, 0.15);  
}
```

```
.card-4 .card-body {  
  padding: 57px 65px;  
  padding-bottom: 65px;  
}
```

```
@media (max-width: 767px) {  
  .card-4 .card-body {  
    padding: 50px 40px;  
  }  
}
```

In templates/users Directory

base.html

```
<!DOCTYPE html>
{%load static%}
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Basepage</title>

    <link rel="stylesheet" href="{ %static 'css/regstyle.css'% }">
  </head>
  <body>
    <div class="reg-page">
      {% block child_block %}
      {% endblock %}
    </div>
  </body>
</html>
```

home.html

```
<!DOCTYPE html>

{% extends 'users/Base.html' %}
{% block child_block %}
{% load widget_tweaks %}

<body>
  <div class="page-wrapper bg-gra-02 p-t-130 p-b-100 font-poppins">
    <div class="wrapper wrapper--w680">
      <div class="card card-4">
        <div class="card-body">
          <h2 class="title">Registration Form</h2>
          <form method="POST">
            <div class="row row-space">
              <div class="col-2">
                <div class="input-group">

                  <label class="label">Name</label>
                  {% render_field form.Name class+="input--style-4"
placeholder="Enter your Name" %}
                </div>
              </div>
              <div class="col-2">
                <div class="input-group">

                  <label class="label">Age</label>
```

```

        {% render_field form.Age class+="input--style-4" placeholder="Enter
your Age" %}
    </div>
</div>
</div>

<div class="row row-space">
    <div class="col-2">
        <div class="input-group">
            <label class="label">Email</label>
            {% render_field form.Email class+="input--style-4"
placeholder="Enter your Email" %}
        </div>
    </div>
    <div class="col-2">
        <div class="input-group">

            <label class="label">Phone Number</label>
            {% render_field form.Mob class+="input--style-4" placeholder="Enter
your Phone number" %}
        </div>
    </div>
    <div class="col-2">
        <div class="input-group">

            <label class="label">Address</label>

            {% render_field form.Add class+="input--style-4" placeholder="Enter
your Address" %}
        </div>
    </div>
</div>
<div class="row row-space">
    <div class="col-2">
        <div class="input-group">

            <label class="label">Password</label>
            {% render_field form.Pass class+="input--style-4" placeholder="Enter
your Password" %}
        </div>
    </div>
    <div class="col-2">
        <div class="input-group">
            <label class="label">Confirm Password</label>
            {% render_field form.Re_Pass class+="input--style-4"
placeholder="Re-enter your Password" %}
        </div>
    </div>
</div>

```

```

        {% csrf_token %}
        <button class="btn btn--radius-2 btn--blue"
type="submit">Submit</button>
    </div>
</form>
</div>
</div>
</div>
</div>

```

```
</body>
```

```
{% endblock % }
```

login.html

```

<!DOCTYPE html>
{% load static %}
<html>
<head>
    <title>Login Page</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
    <link rel="stylesheet" href="{ %static 'css/logstyle.css'% }">
</head>

<body>
    <div class="login-page">
        {% block child_block %}
        {% endblock %}
    </div>

</body>
</html>

```

loginbase.html

```

<!DOCTYPE html>
{% extends 'users/login.html' %}
{% block child_block %}
<div class="form">
    <form method = 'post'>
        {{ log_form.as_p }}
        {% csrf_token %}

```



```

        <button type="login" class="btn btn-primary" >login</button>
        <p class="message">Not registered? <a href="/submit">Create an account</a></p>
    </form>
</div>
{% endblock %}

```

In templates/work Directory

upload.html

```

{% extends 'work/uploadbase.html' %}
{% block child_block %}
<nav class="navbar navbar-dark bg-dark" >
    <span class="navbar-brand">
        ECDSA File Digital Signing
    </span>
    <span class="navbar-brand mb-0 h1">Welcome {{ a }} </span>
</nav>
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <div class="input-group mb-3">
        <div class="custom-file">
            <input type="file" class="custom-file-input" id="inputGroupFile02"
name="document">
            <label class="custom-file-label" for="inputGroupFile02" aria-
describedby="inputGroupFileAddon02">Choose file</label>
        </div>
    </div>
    <button type="submit" class="btn btn-secondary" name="sbutton">Upload File</button>
    <button type="submit" class="btn btn-info" name="ebutton">Encrypt</button>
    <button type="submit" class="btn btn-success" name="vbutton">Verify</button>
    <button type="submit" class="btn btn-outline-danger" name="lbutton">LogOut</button>
</form>

{% if url %}
<p>Uploaded File: <a href="{{ url }}">{{ url }}</a></p>
{% endif %}
{% endblock %}

```

uploadbase.html

```

<!DOCTYPE html>
{% load static %}
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Upload</title>
    </head>

```

```

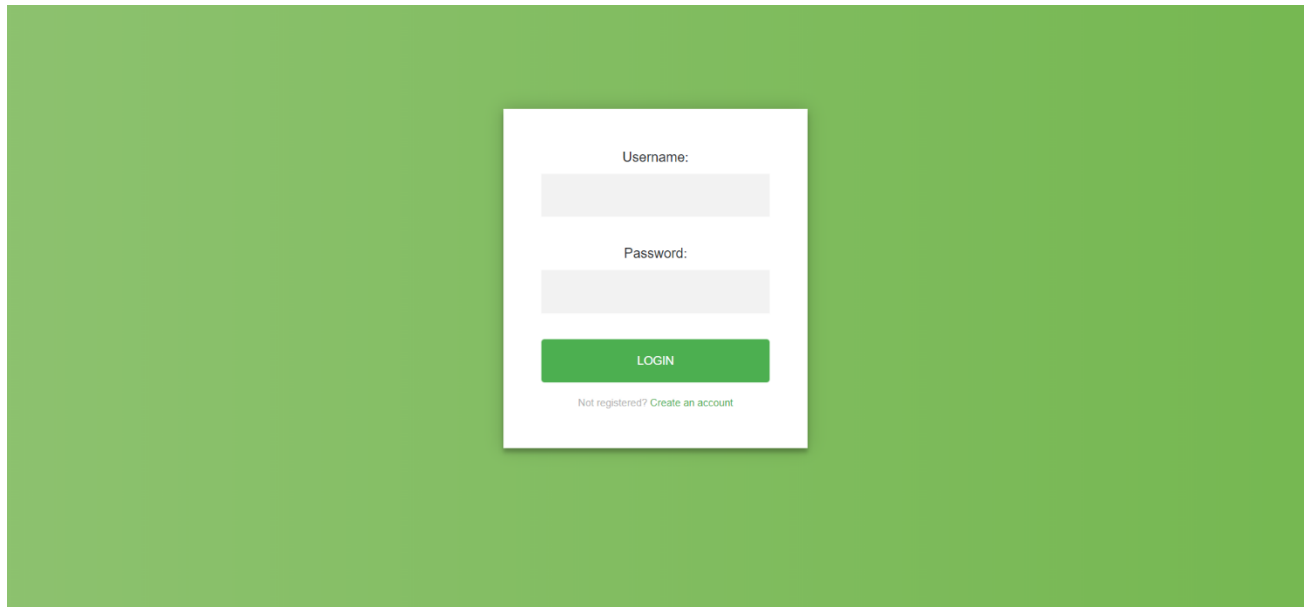
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
<script>
    $(document).ready(function() {
        // messages timeout for 10 sec
        setTimeout(function() {
            $('.msg').fadeOut('slow');
        }, 5000); // <-- time in milliseconds, 1000 = 1 sec
    });
</script>
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
</head>
<body style="background-color: #001f3f;">
    {% block child_block %}
    {% endblock %}
    {% if messages %}
        {% for message in messages %}
            <div class="msg">
                <div class="alert alert-success" role="alert">
                    {{ message }}
                </div>
            </div>
        {% endfor %}

    {% endif %}
</body>
</html>

```

Project Output

1. Login Page



A screenshot of a login page with a green background. A white login form is centered, containing fields for Username and Password, a green LOGIN button, and a link for users not registered.

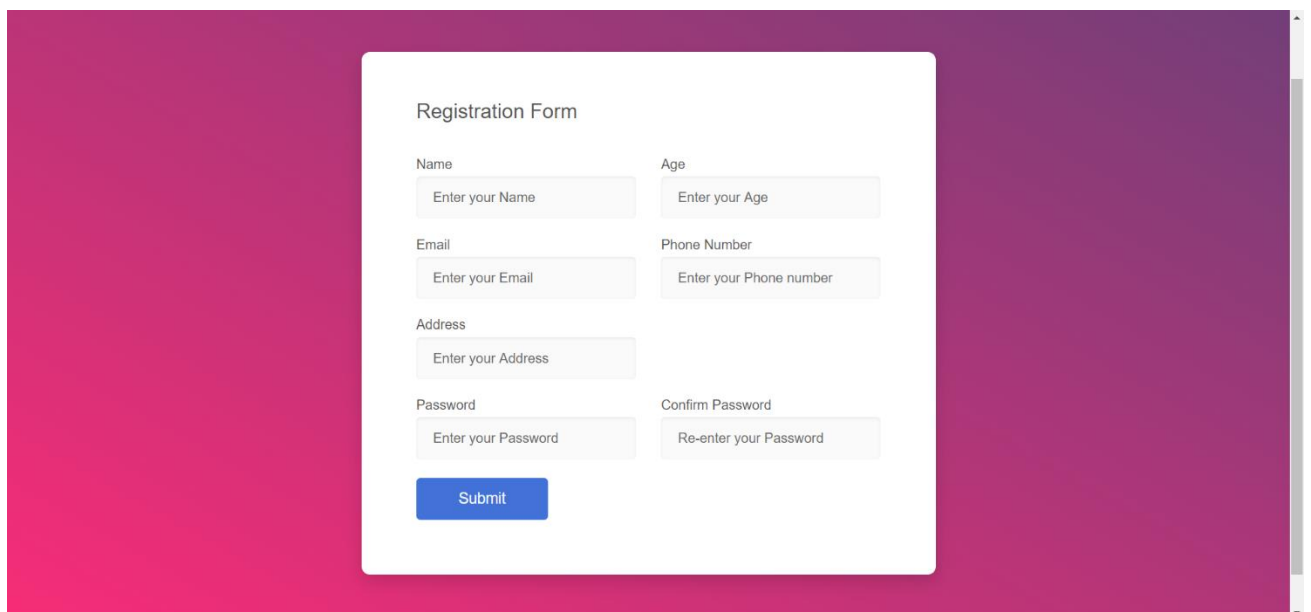
Username:

Password:

LOGIN

Not registered? [Create an account](#)

2. Account Creation Page



A screenshot of an account creation page with a pink-to-purple gradient background. A white registration form is centered, titled 'Registration Form', with fields for Name, Age, Email, Phone Number, Address, Password, and Confirm Password, followed by a blue Submit button.

Registration Form

Name

Age

Email

Phone Number

Address

Password

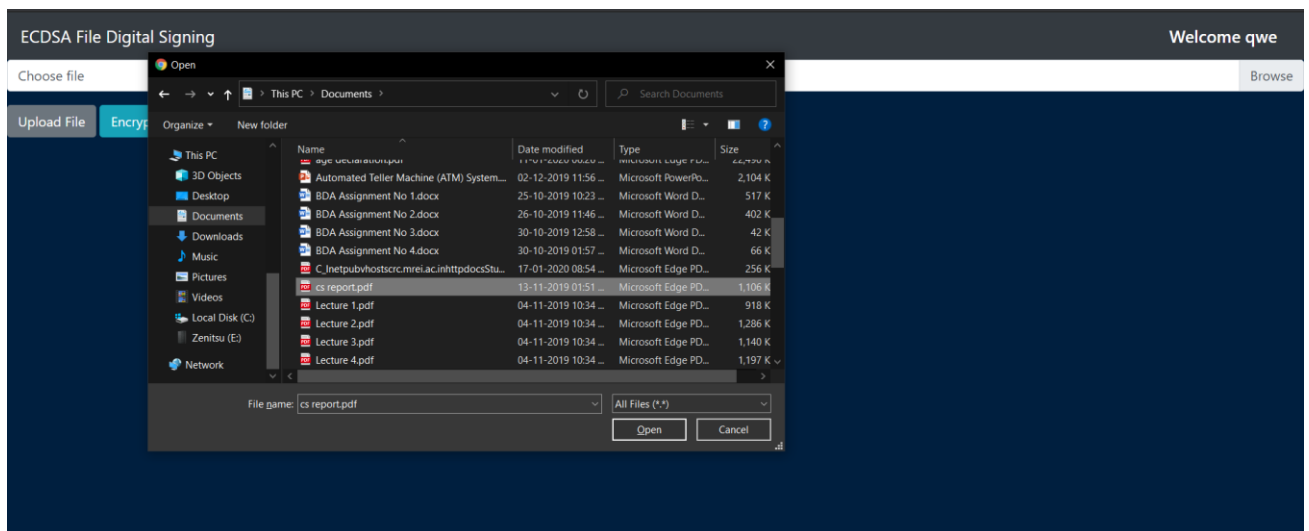
Confirm Password

Submit

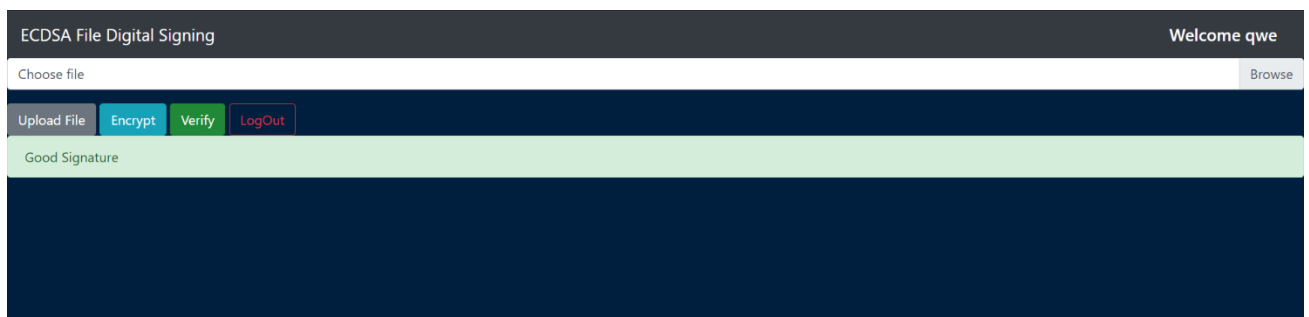
3. Digital Signing Interface Page



4. Selecting Files



5. Verification



Conclusion

5.1 Summary

I studied ECC (Elliptical Curve Cryptography) algorithm. From which I understood and analyzed its functionality and its features. I have optimized its key generation processing speed in an efficient manner. I implemented this ECC project on python run-time environment using ECDSA (Elliptical Curve Digital Signature) library which is a built-in library of python. Also, I designed project's interface using Django and MongoDB. This project is implemented over web interface which includes document digitally signing and all cross verification of signed document. Elliptic curve cryptography is a powerful technology that can enable faster and more secure cryptography across the Internet.

5.2 Future Enhancement

We can use quantum computer's cryptography algorithm which will increase its sophistication and will secure its authentication even more. Also new types of algorithms can be used.

RSA is more secure than ECC as key length is 3072bit keys (which are 50% longer than the 2048bit keys commonly used today).

REFERENCES

- [1] https://www.researchgate.net/publication/268382958_Elliptic_Curve_Digital_Signature_Algorithm_Using_Boolean_Permutation_based_ECC
- [2] <https://www.cs.miami.edu/home/burt/learning/Csc609.142/ecdsa-cert.pdf>
- [3] <https://cryptobook.nakov.com/digital-signatures>