



ENhancing seCurity and privAcy in the Social wEb: a user-centered approach for the protection of minors



WP6 - Sensitive content detection and protection

Deliverable D6.3: "Implementation of browser add-on for content detection and protection"

Editor(s):	Evangelos Kotsifakos, Orfeas Theofanis (LST)
Author(s):	Antonis Papasavva, Petros Papagiannis, Aimilios Yiallourous (CUT), Rig Das, Himanka Kalita, Gabriel Emile Hine, Emanuella Piciucco (ROMA3), Nikolas Kourtellis (TID), Loizos Koukoumas, Christos Mourouzis, Rafael Costantinou, Dimitris Antoniadis (CYRIC), Andreas Pafitis (SGX), Orfeas Theofanis, Dimitrianos Savva, Evangelos Kotsifakos (LST) (CYR), Aimilios Yiallourous (CUT)
Dissemination Level:	Public
Nature:	Demonstrator
Version:	0.4









PROPRIETARY RIGHTS STATEMENT

This document contains information, which is proprietary to the ENCISE Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the ENCISE consortium.

ENCASE Project Profile

Contract Number	691025
Acronym	ENCASE
Title	ENhancing seCurity and privacy in the Social wEb: a user-centered approach for the protection of minors
Start Date	Jan 1 st , 2016
Duration	48 Months

Partners

 Cyprus University of Technology	Cyprus University of Technology	Cyprus
	University College London	United Kingdom
 ARISTOTLE UNIVERSITY OF THESSALONIKI	Aristotle University	Greece
 ROMA TRE UNIVERSITÀ DEGLI STUDI	Università degli Studi Roma Tre	Italy
 Telefonica Investigación y Desarrollo	Telefonica Investigacion Y Desarrollo SA	Spain
 SignalGenerix ADVANCED SIGNAL SOLUTIONS	SignalGenerix Ltd	Cyprus
	Cyprus Research and Innovation Center, Ltd	Cyprus
	L S Tech	Spain

Document History

AUTHORS

(CUT)	Antonis Papasavva, Petros Papagiannis, Aimilios Yiallourous
(ROMA3)	Rig Das, Himanka Kalita, Gabriel Emile Hine, Emanuella Piciucco
(TID)	Nikolas Kourtellis
(CYRIC)	Loizos Koukoumas, Christos Mourouzis, Rafael Costantinou, Dimitris Antoniadis
(SGX)	Andreas Pafitis
(LST)	Orfeas Theofanis, Dimitrianos Savva, Evangelos Kotsifakos

VERSIONS

Version	Date	Author	Remarks
0.1	06.12.2018	LST	Initial partial draft
0.2	11.12.2018	All Authors	Input from authors - Integrated draft
0.3	20.12.2018	CUT	Revision
0.4	28.12.2018	CUT&LST	Final Version

Executive Summary

Following the work done in Tasks T6.2 and T6.3 that has been described in D6.2, D6.3 is the outcome of the task T6.4 – “Design and implementation of browser add-on that informs users about sensitive content and enables them to protect it”. The objective of this task is the development of the browser add-on that detects sensitive content) and enables the user to protect in a user-friendly way. The functionalities of the add-on will be: a) identify user images or messages or newsfeed posts that may be inappropriate for uploading to an OSN without the proper access restrictions; b) warn the user or his parents about the risks of uploading that sensitive content; c) provide usable controls that enables users or their parents to protect such content either by simply defining restrictive OSN privacy settings, or by using steganography and encryption.

The main point of this deliverable is to demonstrate how the browser add-on implementation works and present the background processing. Links to the videos that demonstrate each component are available in this deliverable.

It is shown that the developed technologies have the desired results on the areas of watermarking and steganography, nudity detection and Attribute-based encryption. Technical details of the various modules are described.

Following a seamless integration the modules described in this deliverable are running on the ENCASE proxy and their results are presented to the user through notifications in the browser add-on.

Table of Contents

Executive Summary	4
List of Figures	6
1. Introduction	7
2. Watermarking and Steganography.....	7
2.1. Tools and Libraries	7
2.2. API Implementation	8
2.3. Unhide Encrypted Data from a Steganoimage	11
2.4. Hide Cropped Encrypted Data from an Image in that same Image	13
2.5. Unhide Cropped Encrypted Data from a Steganoimage	16
2.6. Blurring	18
2.7. Watermarking.....	20
3. Skin and face detection.....	22
3.1. System Architecture Flow	22
3.2. Implementation Stack.....	23
3.3. Demonstration and Use Instructions.....	24
4. Nudity detection	29
4.1. How to use the API	30
5. Sensitive Content Detection	33
5.1. Deployment	33
5.2. Demonstration Description	35
6. Attribute-based encryption	35
6.1. Introduction	35
6.2. Tools and Libraries	36
6.3. API Implementation – Demonstration and Use Instructions.....	36
7. Conclusion.....	39
8. Copyright and Intellectual Property	39

List of Figures

Figure 1 – Original image/RGB/HSV/YCbCr and the end average result	23
Figure 2 –Architecture of the proposed in-browser content analysis filter	24
Figure 3 Navigate through different specs	24
Figure 4 “WP6 - T6.1 – LK” Full Website View	25
Figure 5 Services of Face Detection	26
Figure 6 Upload Image to be tested	27
Figure 7 The call to the service	27
Figure 8 Result Returned	27
Figure 9 The body of Face Recognition service takes as input an address.....	28
Figure 10 Example of what the GET request should return	29
Figure 11 The result of face recognition.....	29
Figure 12 Encrypted Image With Access Policy	37
Figure 13 - Decryption API with User's Private Key	38
Figure 14 Decryption Response	39

1. Introduction

Following the research carried out in the previous tasks, this deliverable presents the developed systems that support the detection of sensitive content in OSNs and the way each related module works. The modules are integrated in the ENCASE proxy and are communicating their outputs to the end user through notifications in the browser add-on.

The following implementations are discussed: steganography and digital watermarking, skin and face detection, nudity detection and attribute-based encryption. More on the principles behind the implementation and the related research can be found in the deliverables D6.1 and D6.2.

Demonstrations are available and links to the respective videos are included in the document.

2. Watermarking and Steganography

This chapter reports the Steganography and Digital Watermarking API implementation based on the requirements of ENCASE. The following API calls were developed:

- **Encrypted Steganography in an unsuspected image** - Hide encrypted data (image or text) in a static, unsuspected carrier image (lock)
- **Decrypted De-steganography from a Steganoimage** - Unhide encrypted data (image or text) from a carrier image
- **Cropped Encrypted Steganography** - Crop and blur sensitive content from an image and encrypt and hide the cropped data in that same image
- **Cropped Decrypted De-Steganography** - Decrypt and replace the blurred sensitive content of an image with the cropped data
- **Blur** - Blur sensitive content in an image
- **Watermaking** - Place a watermark of ENCASE logo in an image

The service is up and running on the ENCASE Proxy VM. It can be tested via its Swagger Documentation at <http://35.205.100.70:18080/> or by an API development framework (e.g. [POSTMAN](#)) by calling the relative APIs. Also, a detailed documentation of the implementation and how to install it and run it locally or on the ENCASE Proxy VM can be found in the Phabricator Task Manifest at <http://encase.phabricator.cut.ac.cy/T26>.

In the following sections the implemented API calls are briefly described with step by step screenshots.

2.1. Tools and Libraries

The APIs were developed using:

- [Python 3](#) - Programming Language
- [Stegano](#) - Steganography Library
- [Cryptosteganography](#) - Cryptosteganography Library

- [PyCryptoDome](#) - Python Library for Cryptography
- [PIL](#) - Python Imaging Library
- [Flask](#) - Microframework for web development

2.2. API Implementation

This section reports in brief each API call implementation and provides a step by step screenshot description. The screenshots were taken by calling the APIs through Swagger Documentation of this service which can be found at <http://35.205.100.70:18080/> (WP6 – T6.2- CM).

2.2.1. Encrypted Steganography in an Unsuspected Image

This API call is responsible for hiding encrypted sensitive content in a static, unsuspected image (lock) using LSB Image steganography. It receives as inputs, the content to be hidden (image or text) and the symmetric key to encrypt it and outputs the Steganoimage or an error.

In the case of hiding an image, firstly the algorithm resizes -the bigger dimension to the half of the carrier image and also keeping the aspect ratio- and converts (compress) it to JPEG format in order to ensure that it can fit in the carrier image. Then, the sensitive content (either image or text) is converted in bytes and is encrypted with the symmetric key for extra security before it is finally hidden in the carrier image.

2.2.2. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/hideData
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body (Inputs)

Key	Value	Format
image	Image to be hidden	PNG, JPEG, GIF, BMP
text	Text to be hidden	text
key	Symmetric key for encryption	text

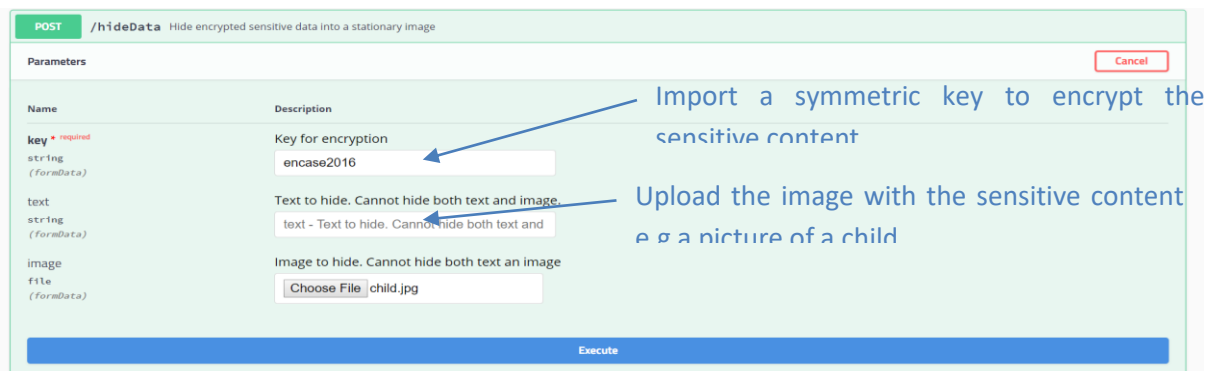
Note: It is not possible to hide both an image and text. Only use one input at a time.

Responses (Outputs)

Status	Value	Description
200	response.png	Steganoiimage containing the hidden data
400	{"error": "Cannot use stegonagraphy without data"}	The API was called with no inputs
400	{"error": "Cannot encrypt data without key"}	The API was called with no key
400	{"error": "Cannot encrypt data with null key"}	The key is null
400	{"error": "File not allowed"}	Input file is not an image or in a supported format
400	{"error": "Cannot import both image and text"}	The API was called with both image and text as inputs

2.2.3. Demonstration and Examples

2.2.3.1. Hide an Image

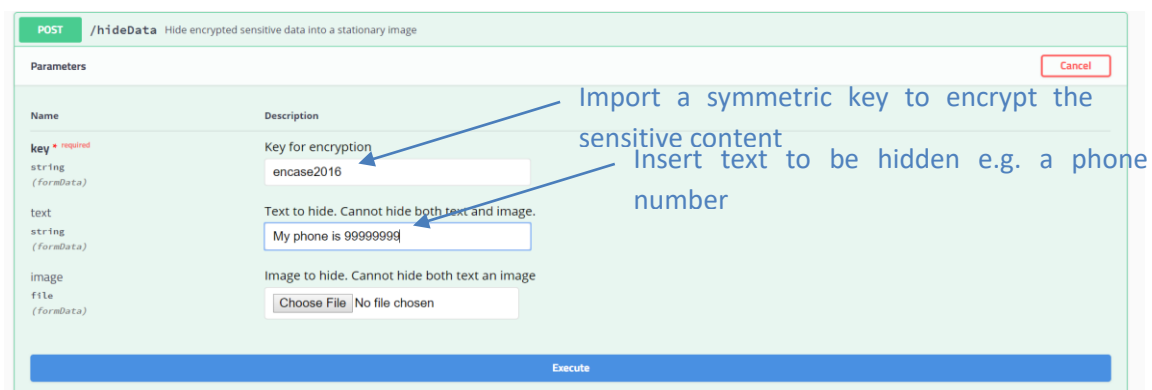


The screenshot shows the 'hideData' API interface with the following parameters:

- key** (required, string, formData): Key for encryption. Value: encase2016. *Annotation: Import a symmetric key to encrypt the sensitive content*
- text** (string, formData): Text to hide. Cannot hide both text and image. Value: text - Text to hide. Cannot hide both text and *Annotation: Upload the image with the sensitive content e.g a picture of a child*
- image** (file, formData): Image to hide. Cannot hide both text an image. Value: Choose File | child.jpg

Buttons: Cancel, Execute

2.2.3.2. Hide Text



The screenshot shows the 'hideData' API interface with the following parameters:

- key** (required, string, formData): Key for encryption. Value: encase2016. *Annotation: Import a symmetric key to encrypt the sensitive content*
- text** (string, formData): Text to hide. Cannot hide both text and image. Value: My phone is 99999999. *Annotation: Insert text to be hidden e.g. a phone number*
- image** (file, formData): Image to hide. Cannot hide both text an image. Value: Choose File | No file chosen

Buttons: Cancel, Execute

2.2.3.3. Response

Curl


```
curl -X POST "http://35.285.109.70:18080/apiSteganographyWatermarking/hideData" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "key=encase2016" -F "text=My phones is 99999999"
```

Request URL

```
http://35.285.109.70:18080/apiSteganographyWatermarking/hideData
```

Server response

Code	Details
200	Response body



SteganolImage that contains
the hidden content (text or
image) in it

2.3. Unhide Encrypted Data from a Steganoimage

This API is responsible for decrypting and revealing the hidden content of a Steganoimage. If the input key is correct and the input image contains hidden content, then the API gives as output the hidden content (either text or image).

2.3.1. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/unhideData
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body (Inputs)

Key	Value	Format
image	Steganoimage that contains the hidden content	PNG
key	Symmetric key for decryption	text

Responses (Outputs)

Status	Value	Description
200	response.jpg	Revealed Image
200	{"hidden": revealed text}	Revealed Text
400	{"error": "Cannot decrypt empty data"}	The API was called with no inputs
400	{"error": "Cannot decrypt data without key"}	The API was called with no key
400	{"error": "Cannot decrypt data with null key"}	The key is null
400	{"error": "File not allowed"}	Input file is not an image or in a supported format
400	{"error": "Nothing is hidden in this image"}	The image does not contain any hidden content

2.3.2. Demonstration and Examples

2.3.2.1. Unhide an Image

POST /unhideData Reveal encrypted sensitive data from a stationary image

Parameters Cancel

Name	Description
key * required string (formData)	Key for decryption <input type="text" value="encase2016"/>
image * required file (formData)	SteganolImage that has hidden content in it. <input type="button" value="Choose File"/> response_hidden_image.png

Import the symmetric key to decrypt the sensitive content
Upload the image that carries the hidden image

2.3.2.2. Response

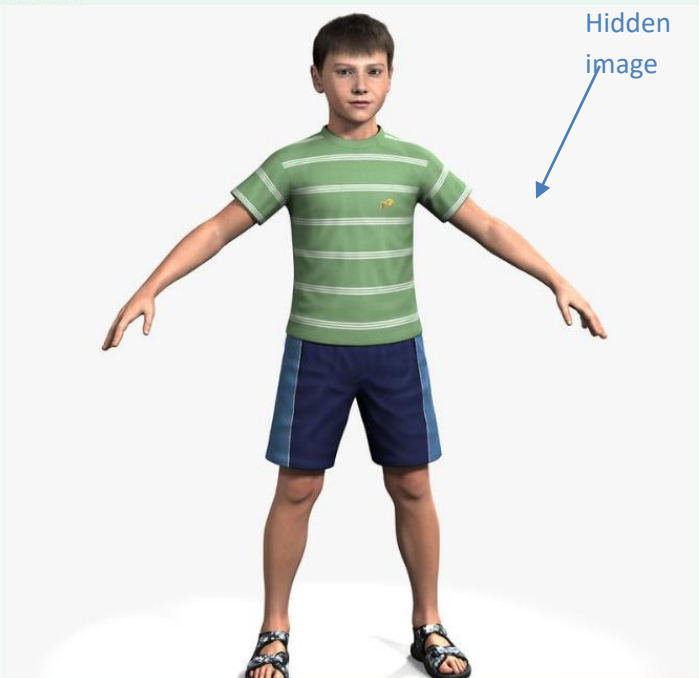
Curl

```
curl -X POST "http://55.205.100.70:10080/apiSteganographyWatermarking/unhideData" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "key=encase2016" -F "image=@response_hidden_image.png;type=image/png"
```

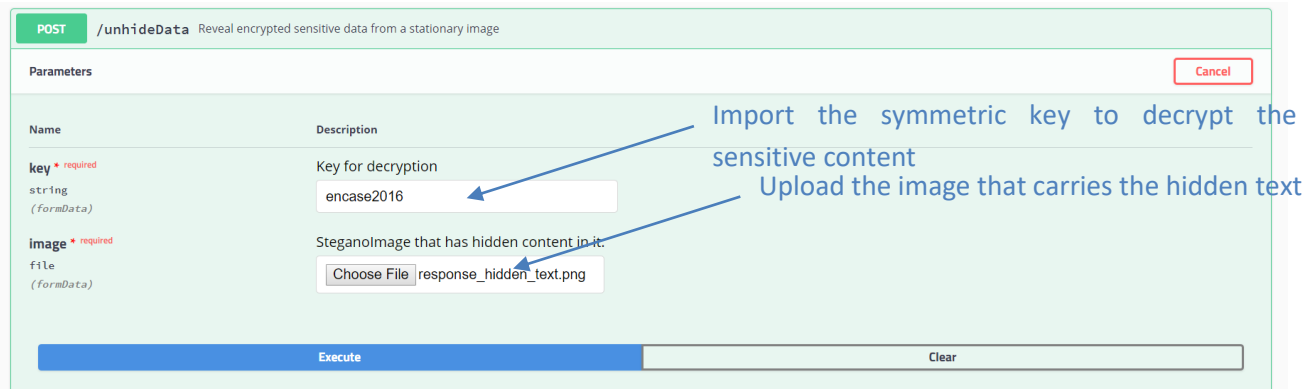
Request URL

```
http://55.205.100.70:10080/apiSteganographyWatermarking/unhideData
```

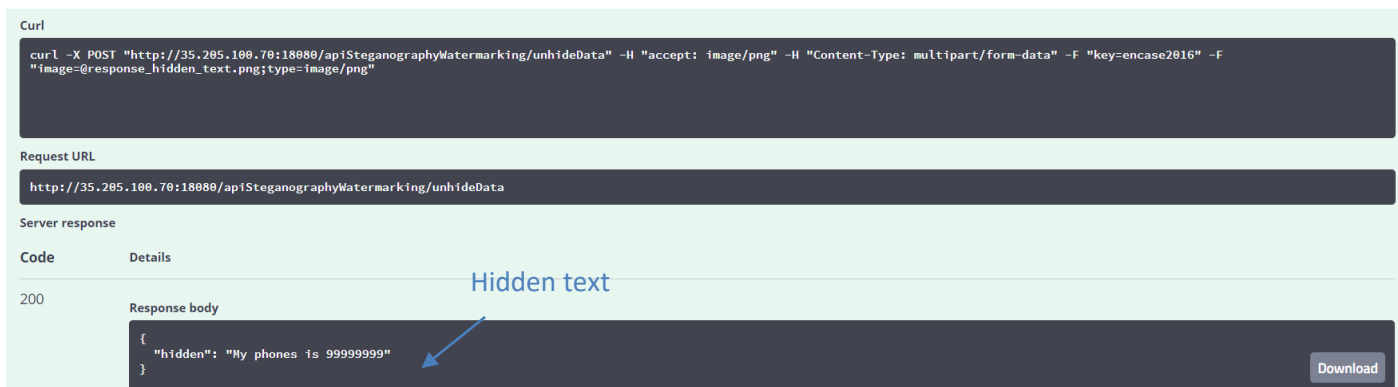
Server response

Code	Details
200	<p>Response body</p> 

2.3.2.3. Unhide Text



2.3.2.4. Response



```
curl -X POST "http://35.285.100.70:18080/apiSteganographyWatermarking/unhideData" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "key=encase2016" -F "image=@response_hidden_text.png;type=image/png"
```

Request URL
http://35.285.100.70:18080/apiSteganographyWatermarking/unhideData

Server response

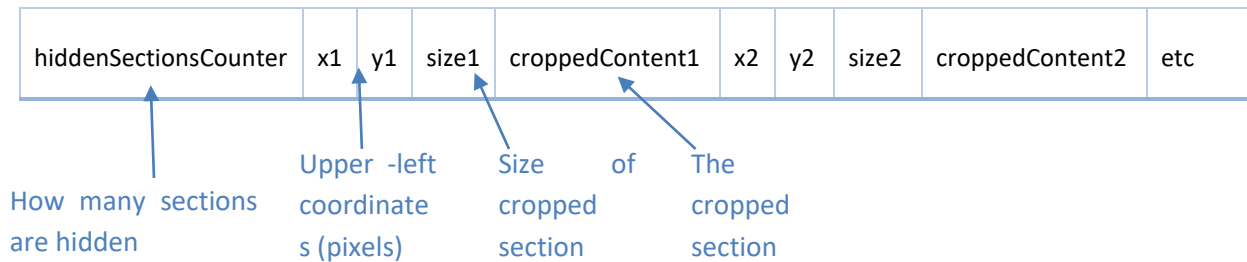
Code	Details
200	<p>Response body</p> <pre>{ "hidden": "My phones is 99999999" }</pre> <p style="text-align: right;">Download</p>

2.4. Hide Cropped Encrypted Data from an Image in that same Image

This API is responsible for cropping and blurring sensitive content of an image, encrypt it with a symmetric key and hide it into the same image using LSB Image steganography. It receives as inputs, the image, the key and the coordinates (pixels) of the sensitive content and outputs the Steganoimage with the sensitive content being blurred or an error.

It is possible to hide multiple sections of an image. This is done by also storing and hiding the upper-left coordinates and the size of each section so that the retrieval algorithm can detect where to replace the blurred section with the original one. The whole content is converted in bytes and is encrypted with the symmetric key for extra security before it is finally hidden in the image.

The format of the hidden content in bytes is:



2.4.1. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/hideCroppedEncryptedData
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body (Inputs)

Key	Value	Format
image	Image with sensitive data that will carry its cropped, encrypted content	PNG, JPEG, GIF, BMP
key	Symmetric key for encryption	text
coordinates	{"coordinates" : [[x1,x2,y1,y2], [x1,x2,y1,y2], etc]}	JSON

Responses (Outputs)

Status	Value	Description
200	response.png	Steganoimage containing the hidden data
400	{"error": "Cannot use steganography without image or coordinates"}	The API was called with no inputs
400	{"error": "File not allowed"}	Input file is not an image or in a supported format
400	{"error": "Cannot encrypt data without key"}	The API was called with

		no key
400	{"error": "Cannot encrypt data with null key"}	The key is null
400	{"error": "Not valid coordinates json. Please send as {'coordinates':[[x1,x2,y1,y2],[x1,x2,y1,y2]]} etc"}	The coordinates value is not a JSON
400	{"error": "Wrong coordinates json format. Please send as {'coordinates':[[x1,x2,y1,y2],[x1,x2,y1,y2]]} etc"}	The coordinates value is not in the correct JSON format

2.4.2. Demonstration and Examples

2.4.2.1. Hide a face

POST /hideCroppedEncryptedData Crop and blur sensitive content from an image and encrypt and hide the cropped data into the image

Cancel

Name	Description	
key * required string <small>(formData)</small>	Key for encryption <input type="text" value="encase2016"/>	Import the symmetric key to encrypt the sensitive content
coordinates * required string <small>(formData)</small>	Coordinates (pixels) [x1, x2, y1, y2] of the sensitive content <input type="text" value='{"coordinates": [[375,440,60,125]]}'/>	Coordinates in pixels of the sensitive section e.g. detected child's face section
image * required file <small>(formData)</small>	Image that has the sensitive content <input type="button" value="Choose File"/> child.jpg	Upload the image that has the sensitive content

Execute

2.4.2.2. Response

```


Curl
curl -X POST "http://35.205.100.70:18080/apiSteganographyWatermarking/hideCroppedEncryptedData" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "key=encase2016" -F "coordinates=[\"coordinates\": [[375,440,60,125]]]" -F "image=@child.jpg;type=image/jpeg"
  
```

Request URL

<http://35.205.100.70:18080/apiSteganographyWatermarking/hideCroppedEncryptedData>

Server response

Code	Details
200	Response body



The child's face is encrypted and hidden inside the image

Output blurred image that carries the cropped encrypted content

2.5. Unhide Cropped Encrypted Data from a Steganoimage

This API is responsible for revealing the hidden, cropped, encrypted content of an image. If the input key is correct and the input image contains hidden content with the correct format, then this content is decrypted and pasted onto its original position in the image.

2.5.1. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/unhideCroppedEncryptedData
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body (Inputs)

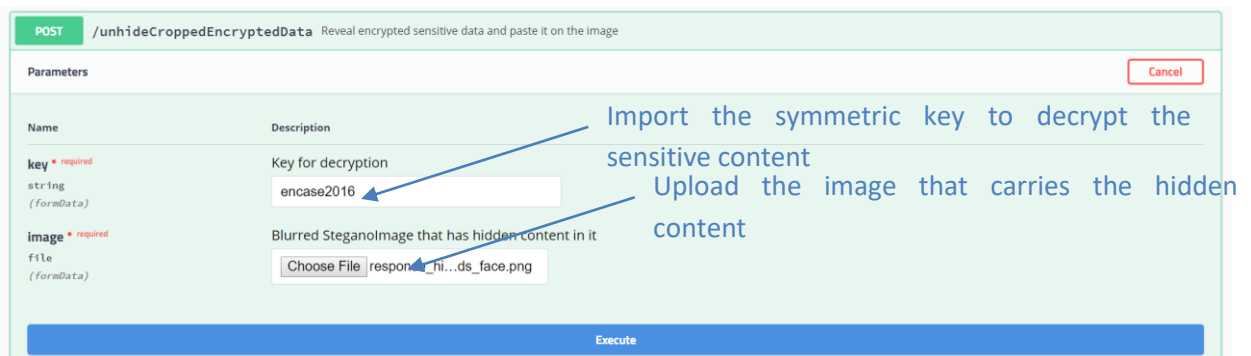
Key	Value	Format
image	Steganoimage that contains the hidden, cropped, encrypted content	PNG
key	Symmetric key for decryption	text

Responses (Outputs)

Status	Value	Description
200	response.jpg	Revealed Image
400	{"error": "Cannot decrypt empty data"}	The API was called with no inputs
400	{"error": "File not allowed"}	Input file is not an image or in a supported format
400	{"error": "Cannot decrypt data without key"}	The API was called with no key
400	{"error": "Cannot decrypt data with null key"}	The key is null
400	{"error": "Nothing is hidden in this image"}	The image does not contain any hidden content

2.5.2. Demonstration and Examples

2.5.2.1. Unhide a face



2.5.2.2. Response

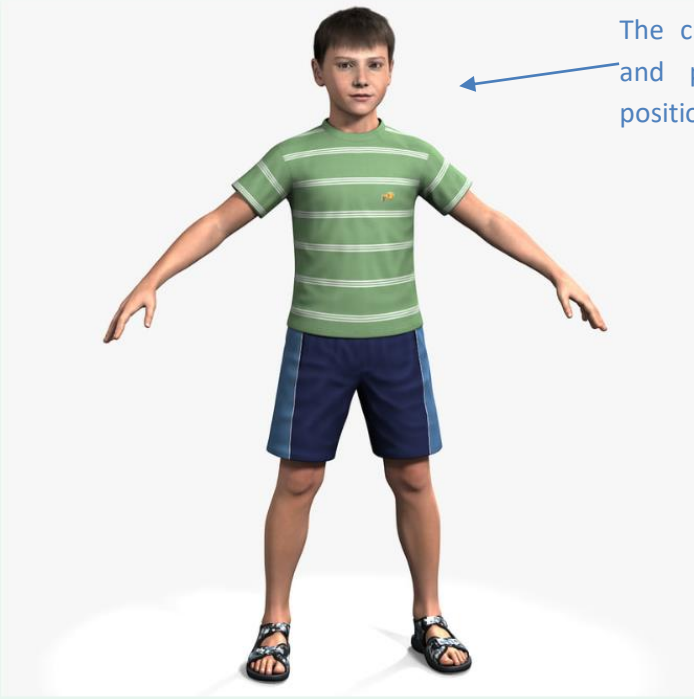
Curl

```
curl -X POST "http://35.205.100.70:18080/apiSteganographyWatermarking/unhideCroppedEncryptedData" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "key=encase2016" -F "image=@response_hidden_childs_face.png;type=image/png"
```

Request URL

```
http://35.205.100.70:18080/apiSteganographyWatermarking/unhideCroppedEncryptedData
```

Server response

Code	Details
200	<p>Response body</p> 

2.6. Blurring

This API is responsible for blurring sensitive content in an image, according to the coordinates (x1, x2, y1, y2) of sensitive content included in that image. The input is an image and a JSON string containing the coordinates (as a JSON array) that indicate the position-s of the content that is needed to be covered. The output is the image having the sensitive content blurred.

2.6.1. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/blur
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body

Key	Value	Format
image	Image to be blurred	PNG, JPEG, GIF, BMP
coordinates	{"coordinates" : [[x1,x2,y1,y2], [x1,x2,y1,y2], etc]}	JSON

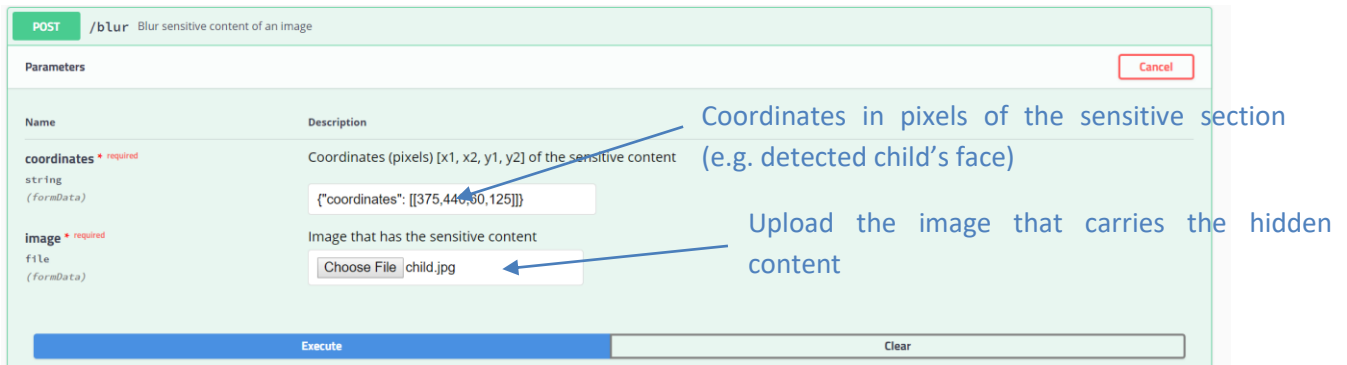
Note: It is possible to cover multi-sections of an image by providing multiple coordinates (x1,x2,y1,y2) with the correct JSON format.

Responses

Status	Value	Description
200	response.png	Image with blurred content
400	{"error": "Cannot blur empty data"}	The API was called without an image as input
400	{"error": "File not allowed"}	Input file is not an image or in a supported format
400	{"error": "Cannot blur without coordinates"}	The API was called without coordinates JSON as input
400	{"error": "Not valid coordinates json. Please send as {'coordinates':[[x1,x2,y1,y2],[x1,x2,y1,y2]]} etc"}	The coordinates value is not a JSON
400	{"error": "Wrong coordinates json format. Please send as {'coordinates':[[x1,x2,y1,y2],[x1,x2,y1,y2]]} etc"}	The coordinates value is not in the correct JSON format

2.6.2. Demonstration and Examples

2.6.2.1. Blur a face



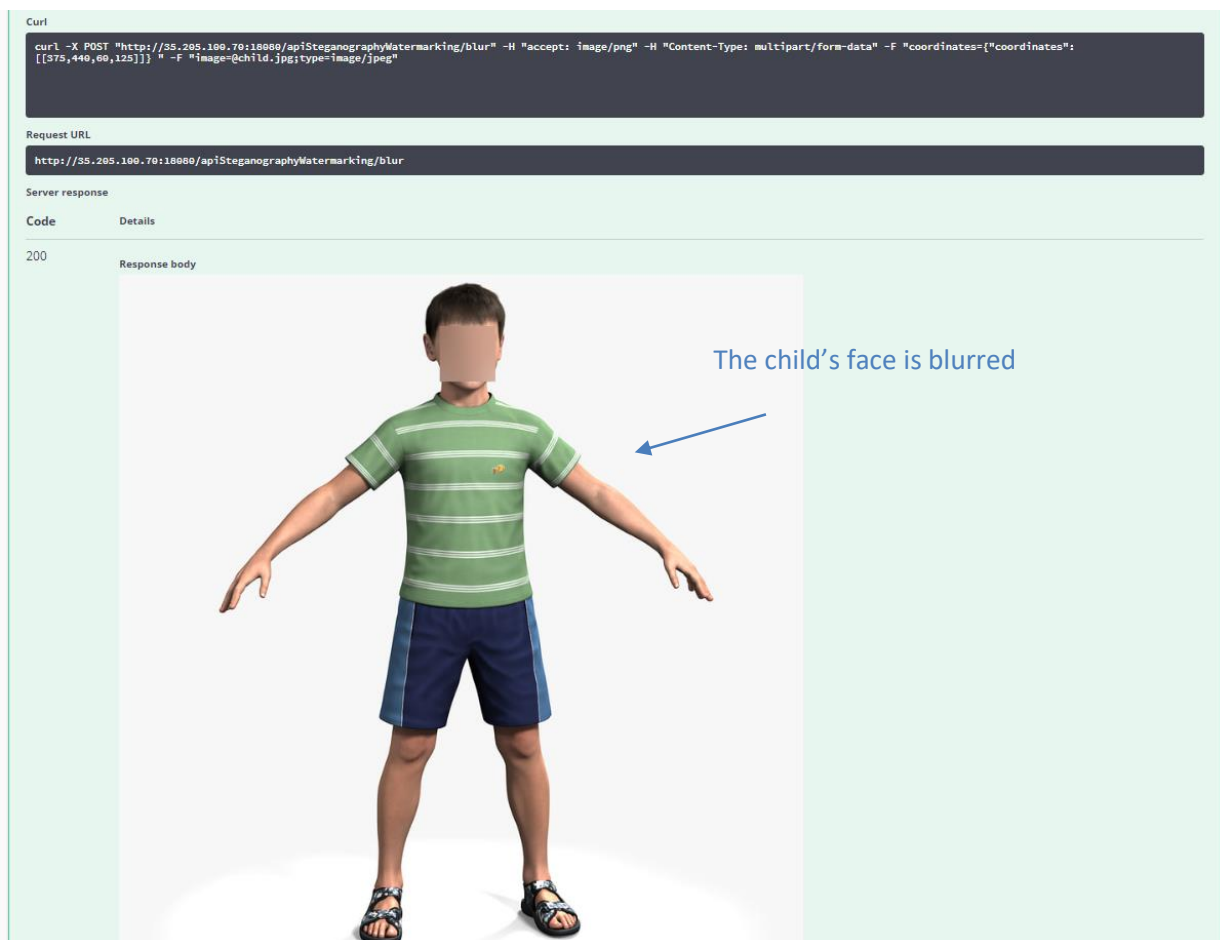
POST /blur Blur sensitive content of an image

Parameters Cancel

Name	Description
coordinates * required string (formData)	Coordinates (pixels) [x1, x2, y1, y2] of the sensitive content (e.g. detected child's face)
image * required file (formData)	Image that has the sensitive content Upload the image that carries the hidden content

Execute Clear

2.6.2.2. Response



Curl

```
curl -X POST "http://35.205.100.70:18080/apiSteganographyWatermarking/blur" -H "accept: image/png" -H "Content-Type: multipart/form-data" -F "coordinates=[[375,440,80,125]]" -F "image=@child.jpg;type=Image/jpeg"
```

Request URL

http://35.205.100.70:18080/apiSteganographyWatermarking/blur

Server response

Code	Details
200	Response body

The child's face is blurred

2.7. Watermarking

This API is responsible for placing a watermark of an ENCAGE Logo in the bottom right of an image. The input is an image and the output is the watermarked image.

2.7.1. Parameters

URL	http://35.205.100.70:18080/apiSteganographyWatermarking/watermark
Method	POST

Header

Key	Value
Content-Type	multipart/form-data

Body (Inputs)

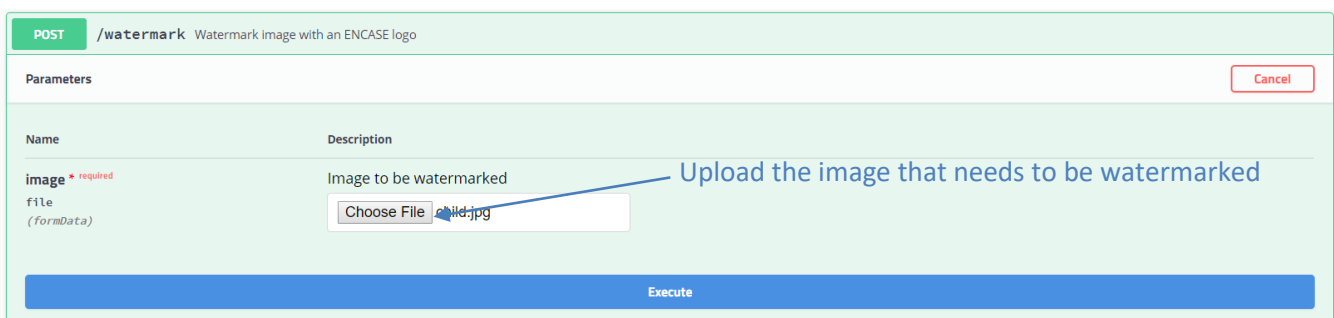
Key	Value	Format
image	Image to be watermarked	PNG, JPEG, GIF, BMP

Responses (Outputs)

Status	Value	Description
200	response.png	Watermarked Image
400	{"error": "Cannot watermark empty data"}	The API was called without an image as input data
400	{"error": "File not allowed"}	Input file is not an image or in a supported format

2.7.2. Demonstration and Examples

2.7.2.1. Watermark an Image



POST /watermark Watermark Image with an ENCASE logo

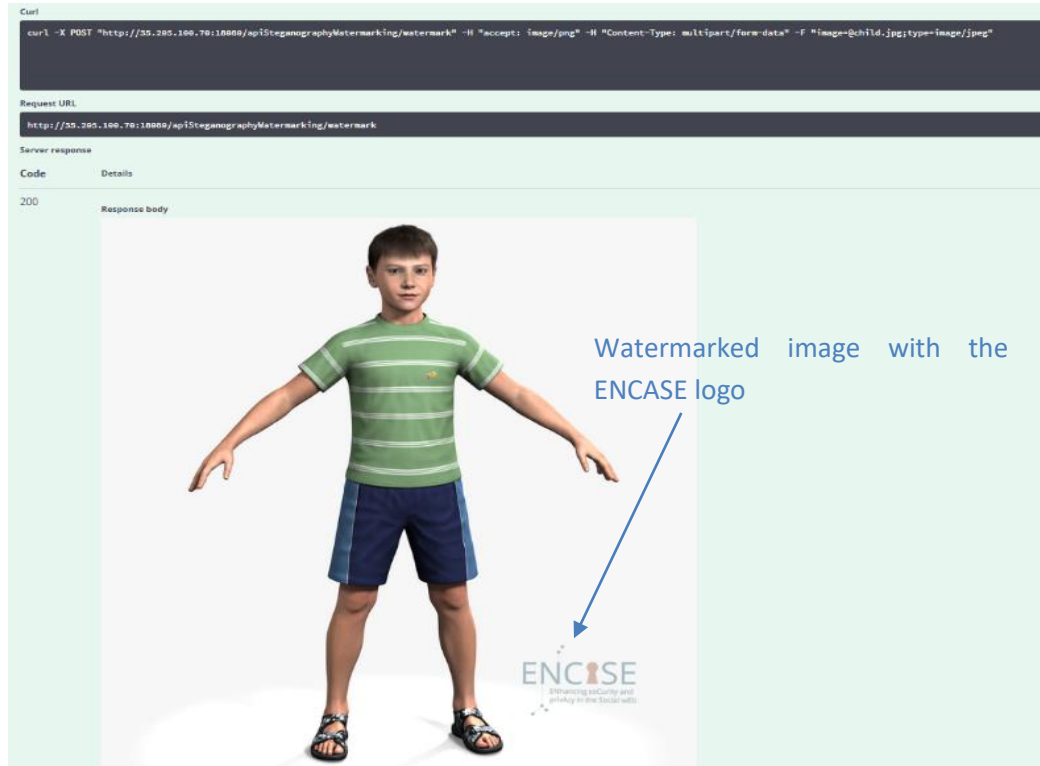
Parameters Cancel

Name	Description
image * required file (formData)	Image to be watermarked

Choose File Upload the image that needs to be watermarked

Execute

2.7.2.2. response



3. Skin and face detection

3.1. System Architecture Flow

This intelligent web proxy depending on the datatype, will decide which module to call. With the call, the intelligent web proxy will also send an execID which indicates which API call is about to be called and a dataID (in case extra data is required) so that the module knows which information is eligible to get.

The current system includes many different and independent modules. Due to the design of the back-end architecture, each module can work independently considering that the right type of parameters is provided. This type of independent modularity gives freedom both to the system and the developers to work using many different programming languages, libraries operating systems. Depending on the API call, the corresponding module will be called. The module will process the request, using the dataID it will retrieve information from the database and return a result (again depending on the call) along with an execID. Using the execID, the intelligent web proxy knows which API call is answered. If found necessary, the data will be stored in the database using an appropriate dataID for future use.

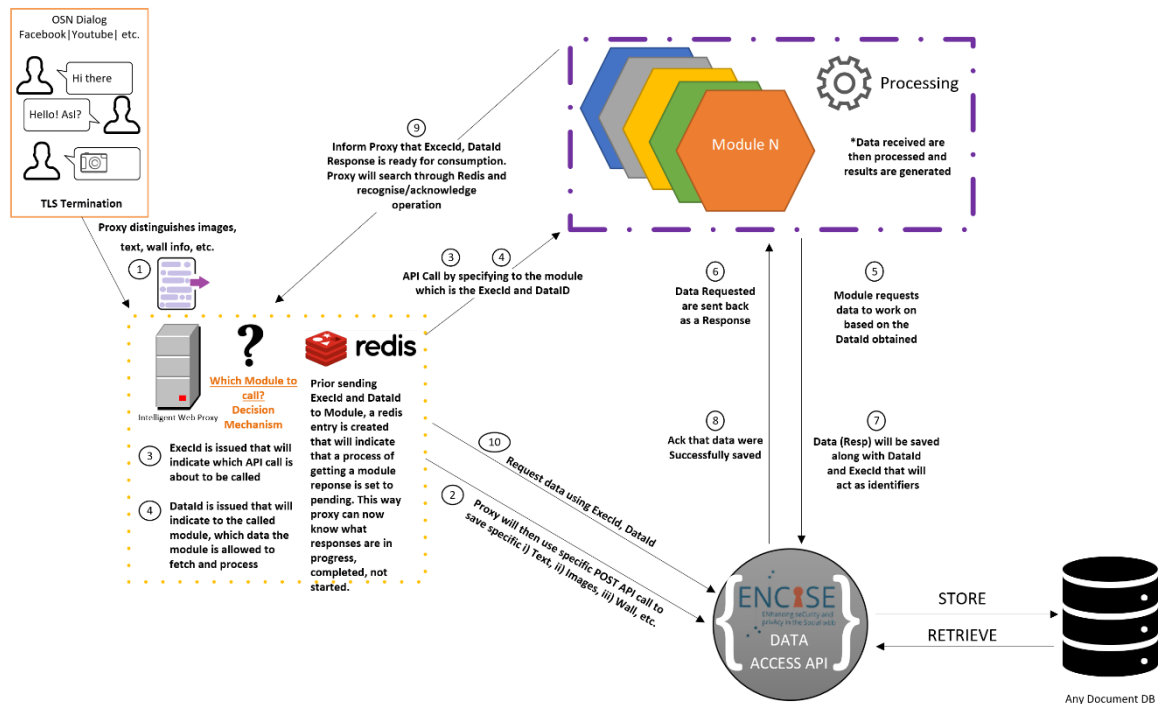


Figure 1 – Original image/RGB/HSV/YCbCr and the end average result

3.2. Implementation Stack

Firstly, we have the front-end where in our case could be the add-on or any other website that requires our API. The website in Collaboration with an API manager (this case Swagger) would make an API request. Our current server runs on NGINX with Gunicorn as an App server. Then for handling the API requests we use Falcon. Depending on the API call, Falcon decides which class to call. The classes are written in Python 2 using OpenCV. A more detailed list of the libraries used can be found in APPENDIX A.

When the algorithms finish processing the data, a result is returned in either JSON format or xml (face recognition training).

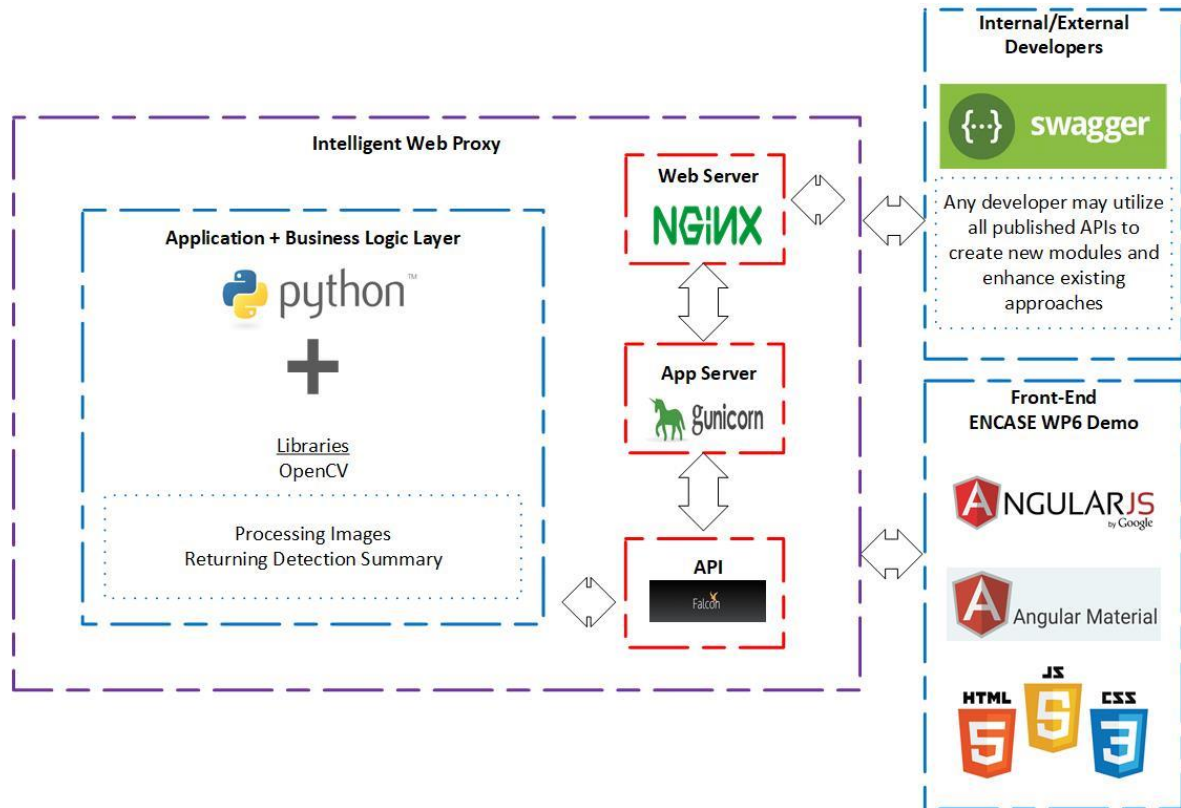


Figure 2 –Architecture of the proposed in-browser content analysis filter

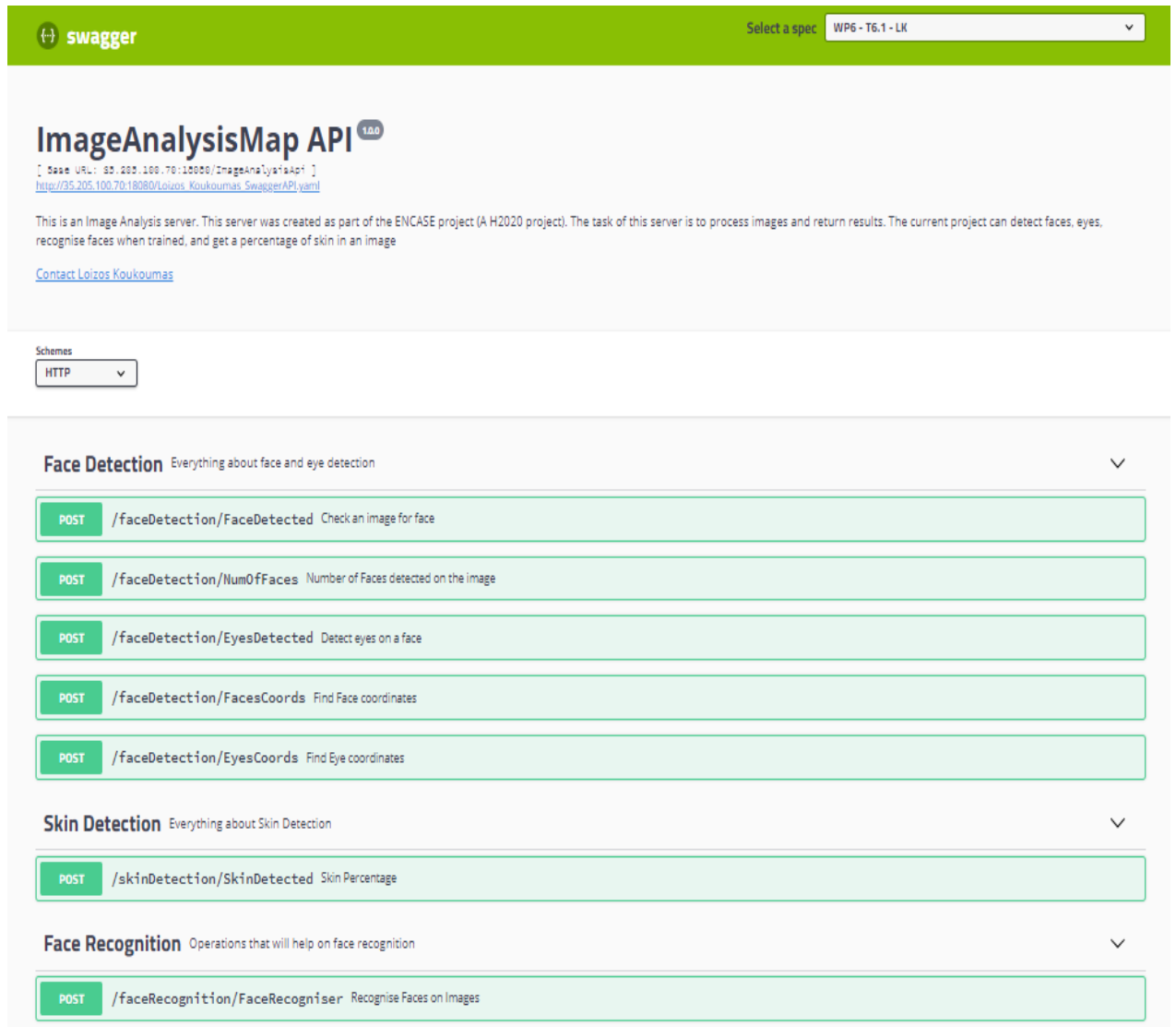
3.3. Demonstration and Use Instructions

The implementation of the algorithms and the services can be demonstrated through the swagger website. Using the front-end server address : <http://35.205.100.70:18080> . From here you can select a specific spec to use. For navigating through the different specs, use the menu toolbar on the top right.



Figure 3 Navigate through different specs

When you navigate on a spec, you can see the corresponding services available. In the situation of “WP6 – T6.1 – LK” the site looks like Figure 4.



swagger Select a spec: WP6 - T6.1 - LK

ImageAnalysisMap API ^{1.0.0}

[Base URL: 53.205.100.70:18080/ImageAnalysisMapAPI]
http://35.205.100.70:18080/Loizos_Koukoumas_SwaggerAPI.yaml

This is an Image Analysis server. This server was created as part of the ENCASE project (A H2020 project). The task of this server is to process images and return results. The current project can detect faces, eyes, recognise faces when trained, and get a percentage of skin in an image

[Contact Loizos Koukoumas](#)

Schemes
 HTTP

Face Detection

Everything about face and eye detection

- POST /faceDetection/FaceDetected Check an image for face
- POST /faceDetection/NumOfFaces Number of Faces detected on the image
- POST /faceDetection/EyesDetected Detect eyes on a face
- POST /faceDetection/FacesCoords Find Face coordinates
- POST /faceDetection/EyesCoords Find Eye coordinates

Skin Detection

Everything about Skin Detection

- POST /skinDetection/SkinDetected Skin Percentage

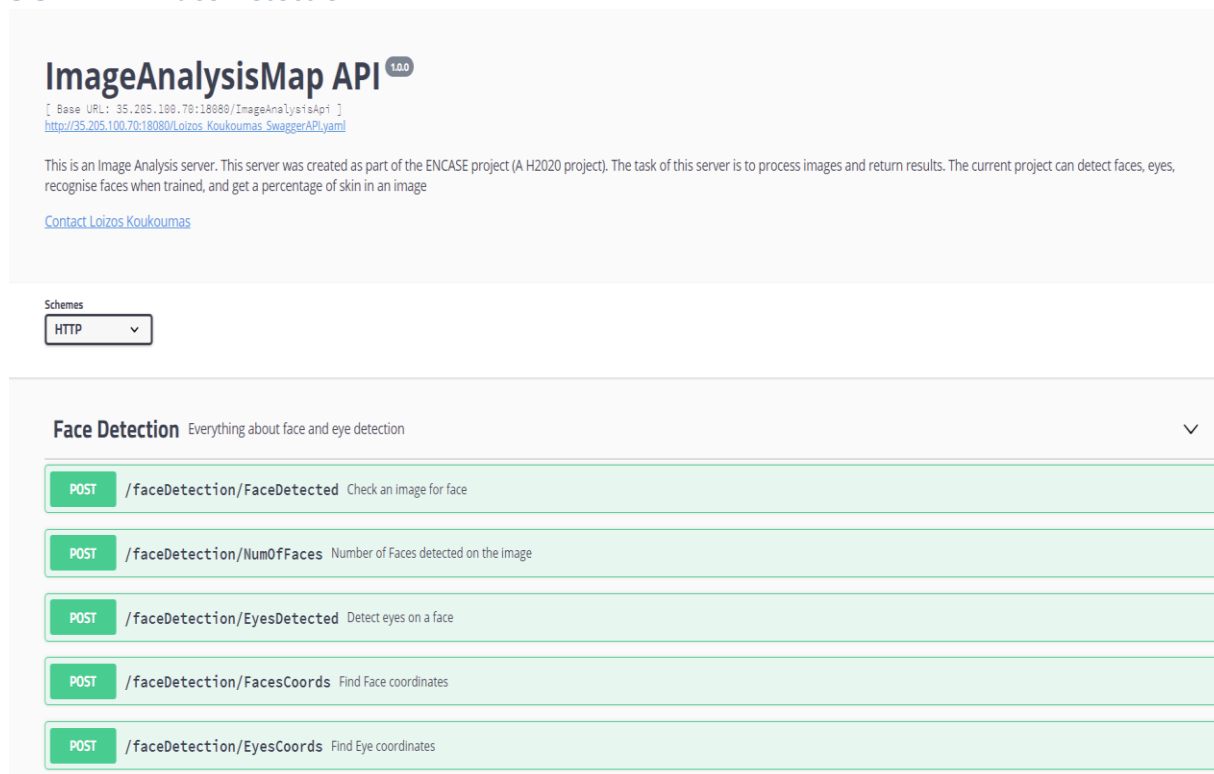
Face Recognition

Operations that will help on face recognition

- POST /faceRecognition/FaceRecogniser Recognise Faces on Images

Figure 4 “WP6 - T6.1 – LK” Full Website View

3.3.1. Face Detection



ImageAnalysisMap API ^{1.0.0}
 [Base URL: 35.205.100.70:18080/ImageAnalysisApi]
<http://35.205.100.70:18080/Loizos.Koukoumas.SwaggerAPI.yaml>

This is an Image Analysis server. This server was created as part of the ENCASE project (A H2020 project). The task of this server is to process images and return results. The current project can detect faces, eyes, recognise faces when trained, and get a percentage of skin in an image

[Contact Loizos Koukoumas](#)

Schemes

Face Detection Everything about face and eye detection

- POST** /faceDetection/FaceDetected Check an image for face
- POST** /faceDetection/NumOfFaces Number of Faces detected on the image
- POST** /faceDetection/EyesDetected Detect eyes on a face
- POST** /faceDetection/FacesCoords Find Face coordinates
- POST** /faceDetection/EyesCoords Find Eye coordinates

Figure 5 Services of Face Detection

Face detection is made following the theory Viola-Jones algorithm and developed using the OpenCV library.

An example of using the services in swagger is demonstrated using screenshots from the actual site. For the POST requests you give a value to the body of the request (Figure 6). In this scenario an image. The image is uploaded on the proxy server. After the image is processed, the result can be viewed in the response body(Figure 8). In case of an error, a different message is displayed. Finally there is the “Curl” screen, where someone can take the actual call and recreate it in a console terminal, using “curl”. All the other calls with one exception follow the same logic. They all take as input an image and return an answer in JSON form. In the next session you can read about the services in more details.

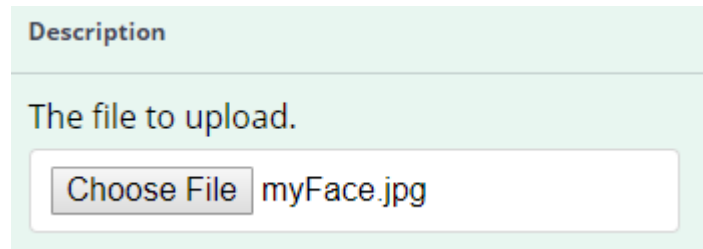


Figure 6 Upload Image to be tested

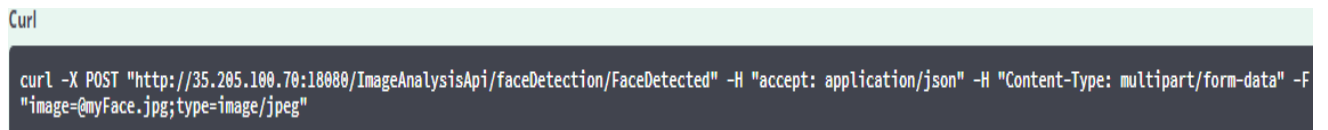


Figure 7 The call to the service

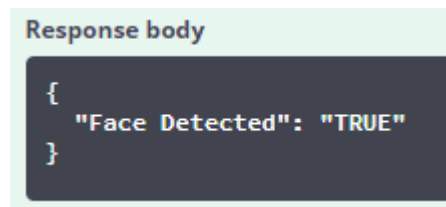


Figure 8 Result Returned

The available API calls for face detection are:

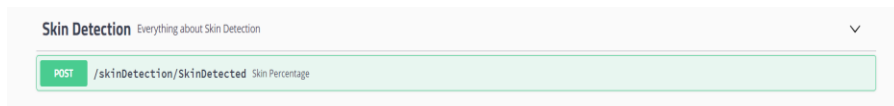
- **/ImageAnalysisApi/faceDetection/FacesDetected:** This call, takes as a parameter an image. The algorithm detects if there are faces in the image and returns a Boolean (True/False)
- **/ImageAnalysisApi/faceDetection/NumOfFaces:** Takes an image as a parameter and returns the number of faces found in the image.
- **/ImageAnalysisApi/faceDetection/FaceCoords:** Takes as a parameter an image, it detects faces and returns their coordinates in the form of x1, x2, y1, y2.
- **/ImageAnalysisApi/faceDetection/EyesDetected:** Takes an image as a parameter and returns a Boolean (true/false) if any eyes were detected.
- **/ImageAnalysisApi/faceDetection/EyesCoords:** Takes an image as a parameter and returns the eyes coordinates in the form of x1, x2, y1, y2.

3.3.2. Skin Detection

The skin detection algorithm was developed using the three filters approach described in section 2.5.3. as well as an approach using the human body detection described in Section 2.5.4

The available API calls for skin detection are:

- **/ImageAnalysisApi/SkinDetection/SkinPercentageInImage:** Takes as an input a picture and returns the percentage of skin detected in the whole image



- **/ImageAnalysisApi/SkinDetection/SkinPercentageOnBody:** Takes as an input an image and returns the percentage of skin detected in different human body parts that are detected, excluding the percentage detected on the face.

3.3.3. Face Recognition

Face recognition was developed using the LBPH method.

The Face Recognition requirements are covered by one service. This service is the only service that doesn’t take as input an image, but instead an address. This address returns valuable information such as the location of the images that the service will need for training for each person in the house, and the location of the image that is to be tested.

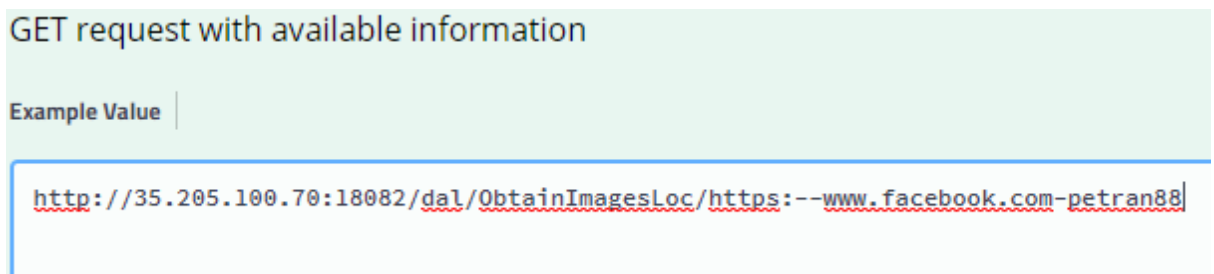


Figure 9 The body of Face Recognition service takes as input an address

The table shows the field that are required to be returned from the service:

Variables	Value Description
_id	This is the ID of the family in the corresponding database
data_id	The ID of the data sent for processing
count_child	The number of person the algorithm will train on
test_loc	The location on the server from where the server can get the image to be tested

child_x	The location of the images of child number x. This folder needs to contain a set of images of a specific person so that the algorithm can be trained on.
---------	--

The next figure shows an example of how the GET result should look like. In this example there are 2 children to be identified.

```
{
  "Images": [
    {
      "_id": {
        "$oid": "5bfd693e417c773c21f9fae9"
      },
      "data_id": "https://www.facebook.com-petran88",
      "count_child": 2,
      "test_loc": "/var/www/html/console/Encase_Parental_Console/uploads/testing/",
      "child_1": "/var/www/html/console/Encase_Parental_Console/uploads/https://www.facebook.com-apapasavva",
      "child_2": "/var/www/html/console/Encase_Parental_Console/uploads/https://www.facebook.com-petran88"
    }
  ]
}
```

Figure 10 Example of what the GET request should return

The service on the proxy will use this data, get the necessary information, train the set, test the image and return the identified person’s name with a value of confidence.

```
[
  "https://www.facebook.com-apapasavva",
  70.27773261537762
],
```

Figure 11 The result of face recognition

4. Nudity detection

This section contains the documentation (Windows) for the implemented API calls for:

- Nudity detection in an image – Estimate the nudity probability in a given image

The network takes in an image and gives output a probability (score between 0-1) which can be used to filter nudity images. Scores < 0.2 indicate that the image is likely to be safe with high

probability. Scores > 0.8 indicate that the image is highly probable to be nude. Scores in middle range may be binned for different nudity levels.

The APIs were developed using:

- [Python 3](#) - Programming Language
- [Numpy](#) – Python package for scientific computing
- `Os` – Python package for operating system dependent functionality
- `Sys` – Python package for System-specific parameters and functions
- [Argparse](#) – Python package for command-line options, arguments and sub-commands
- [Glob](#) - Python package for Unix style pathname pattern expansion
- [Caffe](#) – Python framework for deep learning

4.1. How to use the API

Install the requirements

Make sure you have Python 3 installed. Then, navigate to the directory

path `~/aimilios/nudityDetection/` and run the following command in a terminal:

```
pip install -r requirements.txt
```

Run the API

Run the service using the following command:

```
python .....
```

Call the APIs

We can use the `classify.py` script to run the `nudityDetection` model and it prints the nudity probability score.

```
>>>> Python classify.py INPUT_IMAGE_PATH
```

We will get the nudity score returned:

```
Nudity Probability: 0.975002
```

4.1.1. Demonstration and Examples

Below, are three example images with their respective results:

4.1.1.1. *Example 1*



*Image is blurred because it contains pornographic material.

Result

```
Administrator: Anaconda Prompt
(base) C:\python\nudity>python classify.py 1.jpg
Nudity Probability: 0.975002
(base) C:\python\nudity>
```

Nudity Probability: 0.975002

4.1.1.2. *Example 2*



Result

```
Administrator: Anaconda Prompt
(base) C:\python\nudity>python classify.py 2.jpg
Nudity Probability: 0.000001
(base) C:\python\nudity>
```

Nudity Probability: 0.000001

4.1.1.3. *Example 3*



Nudity Probability: 0.108276

Result

```
Administrator: Anaconda Prompt
(base) C:\python\nudity>python classify.py 3.jpg
Nudity Probability: 0.108276
(base) C:\python\nudity>
```


5. Sensitive Content Detection

For this work, we used some existing libraries¹ and we created scripts that can successfully detect:

- Dates
- Times
- Phone numbers
- Phone numbers with extensions
- Links
- Emails
- IP addresses
- IPv6 addresses
- Prices
- Credit card numbers
- Street Addresses
- Zip codes

5.1. Deployment

This script is deployed in the Intelligent Web-Proxy of our architecture, in port 8585 (35.205.100.70:8585). It offers the following APIs:

5.1.1. Detect Sensitive Content from Post

In this scenario, the API receives as an input a text, and returns a json file of the detected sensitive contents.

Request:

```
GET /apiDetectSensitiveContent/fromPost?text="Hello my address is 1 Georgiou street, AJ1698. I will wait for you at 16:00. If you can't find me, call me at 0035799803340. BTW, I've just bought a new phone worth €500, and I used my credit card 1234569814789632" HTTP/1.1
```

```
Host: 35.205.100.70:8585
```

```
cache-control: no-cache
```

```
Postman-Token: 076eb6b1-7826-4af1-976a-b40c2af60e35
```

Response:

```
{  
  "dates": [],  
  "times": [  
    "16:00"  
  ],  
  "links": [],  
  "phones": [  
    "0035799803340"  
  ],  
  "prices": [  
    "€500"  
  ],  
  "emails": [],  
  "ip_addresses": [],  
  "ipv6_addresses": [],  
  "credit_cards": [  
    "1234569814789632"  
  ],  
  "street_addresses": [  
    "1 Georgiou street, AJ1698"  
  ],  
  "zip_codes": []  
}
```

¹ <https://github.com/madisonmay/CommonRegex>

```
"0035799803340"
],
"ext_phones": [],
"emails": [],
"ips": [],
"prices": [
  "€500"
],
"credit_cards": [
  "1234569814789632"
],
"street_addresses": [
  "1 Georgiou street,"
],
"po_boxes": [],
"zip_codes": []
}
```

5.1.2. Detect Sensitive Content from Chat

In this scenario, the API receives as an input a URL, and returns a json file of the detected sensitive contents.

Request:

```
GET
/apiDetectSensitiveContent/fromChat?url=http://35.205.100.70:18082/dal/ObtainData/chat/date_created/11-12-2018/conv_https:--www.facebook.com-somebaduser.papas.5_https:--www.facebook.com-petran88 HTTP/1.1
Host: 35.205.100.70:8585
cache-control: no-cache
Postman-Token: da17ac73-cf4d-48ce-99eb-e4570ec9d601
```

Response:

```
{
  "dates": [
    "17-12-2018",
    "17-12-2018",
    "11-12-2018"
  ],
  "times": [],
  "links": [
    "--www.facebook.com-somebaduser.pa",
    "--www.facebook.com",
  ]
}
```

```

"--www.facebook.com"
],
"phones": [
  "123886417",
  "12388603"
],
"ext_phones": [],
"emails": [],
"ips": [],
"prices": [],
"credit_cards": [],
"street_addresses": [
  "2 cover mehi richard."
],
"po_boxes": [],
"zip_codes": []
}

```

5.2. Demonstration Description

For this Demonstration, a minor tries to post sensitive content publicly in his Facebook profile. The browser add-on captures this information and sends it to the IWP for analysis. If the IWP detects any sensitive content, then the minor will be notified immediately.

Please visit the link below to watch a video of this demonstration:

<https://www.dropbox.com/s/hgsudwh37lu5z8r/SensitiveContentDetection.mp4?dl=0>

6. Attribute-based encryption

6.1. Introduction

The following document reports the Attribute-based Encryption (Task 6.3) API Implementation based on the requirements of the European Project of Encase. The following API calls were developed for the purpose of Task 6.3 and Task 6.4.

- Encrypt Image using Ciphertext Policy ABE – Hide image by defining an access policy and generates pair of keys for the users.
- Decrypt Image – Unhide Image by using the private key of a user.

In the following sections the implemented API calls are briefly described with step by step screenshots.

6.2. Tools and Libraries

The libraries and tools that were used for the development of the APIs are the following:

- Python 3 - Programming Language
- Charm: A Framework for Rapidly Prototyping Cryptosystems

6.3. API Implementation – Demonstration and Use Instructions

6.3.1. Image Encryption

The following API calls are responsible for the attribute-based encryption scheme. As it was stated in D6.2, the attribute-based encryption has four phases that happening in the background except the encryption and decryption that is happening on the intelligent web proxy. We suppose that the private key is known just for demonstration purposes.

6.3.2. Encryption of Image Demonstration

Parameters:

URL	virtserver.swaggerhub.com/Individual68/T6.3AttributeBasedEncryption/1.0.0/EncryptImage
Method	POST

Header:

Key	Value
Content-Type	multipart/form-data

Inputs:

Key	Value	Format
image	Image to be hidden	JPEG
Access Policy	Access Policy Tree	text

Outputs:

Status	Value	Description
200	Encrypted.jpeg	Encrypted Image
400	{"error": "Cannot use attribute based without input image"}	The API was called with input image
400	{"error": "Cannot encrypt image without access tree"}	The API was called with access policy

6.3.2.1. Demonstration

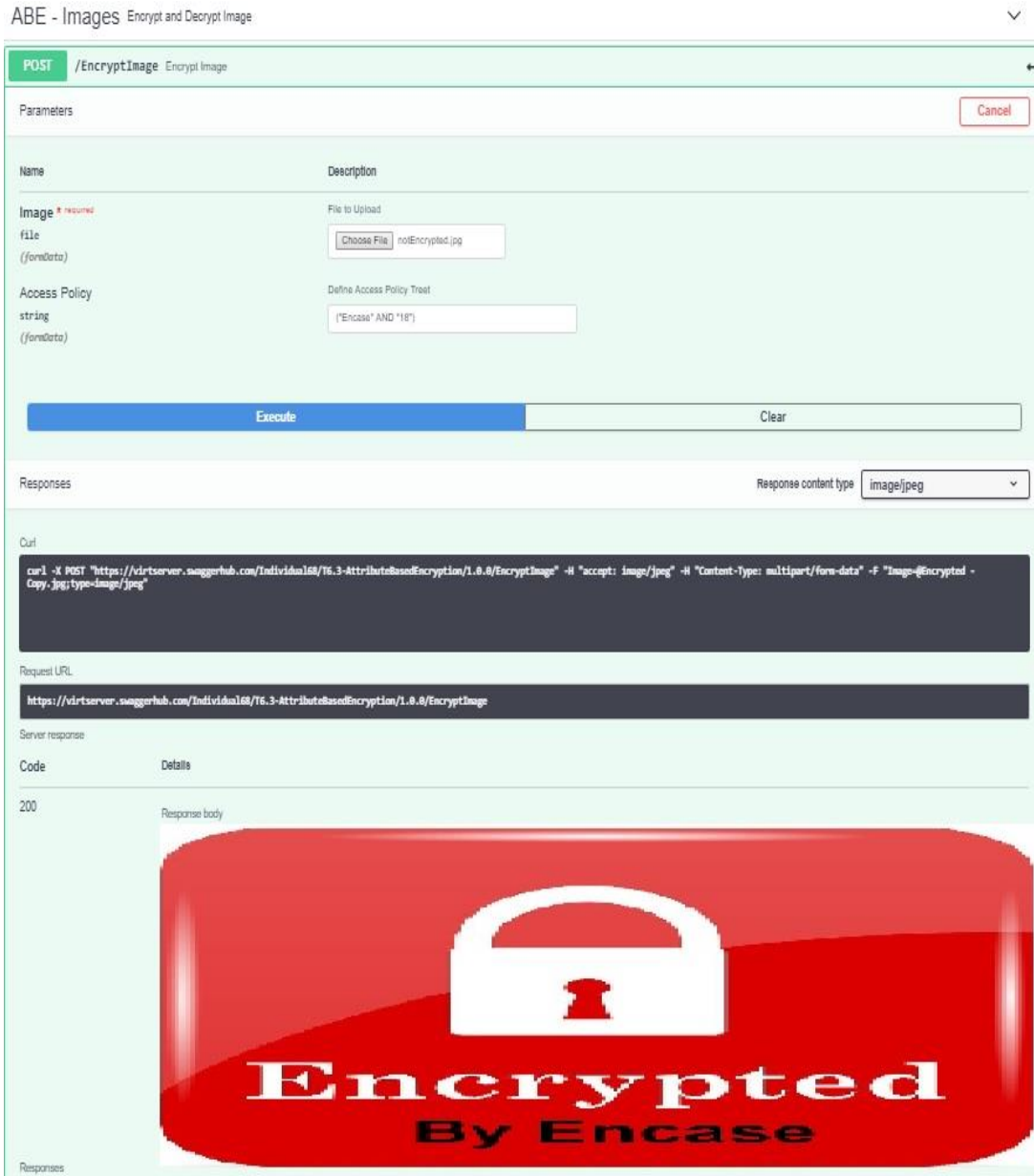


Figure 12 Encrypted Image With Access Policy

6.3.3. Decryption of Image Demonstration

Parameters:

URL	<code>virtserver.swaggerhub.com/Individual68/T6.3AttributeBasedEncryption/1.0.0/DecryptImage</code>
Method	POST

Header:

Key	Value
Content-Type	multipart/form-data

Inputs:

Key	Value	Format
image	Image to decrypt	JPEG
Private Key	User Private Key	alphanumeric

Outputs:

Status	Value	Description
200	notEncrypted.jpeg	Decrypted Image
400	{"error": "Cannot decrypt attribute based without input image"}	The API was called with input image
400	{"error": "Cannot decrypt image without access tree"}	The API was called without entering private key.

6.3.3.1. Demonstration:

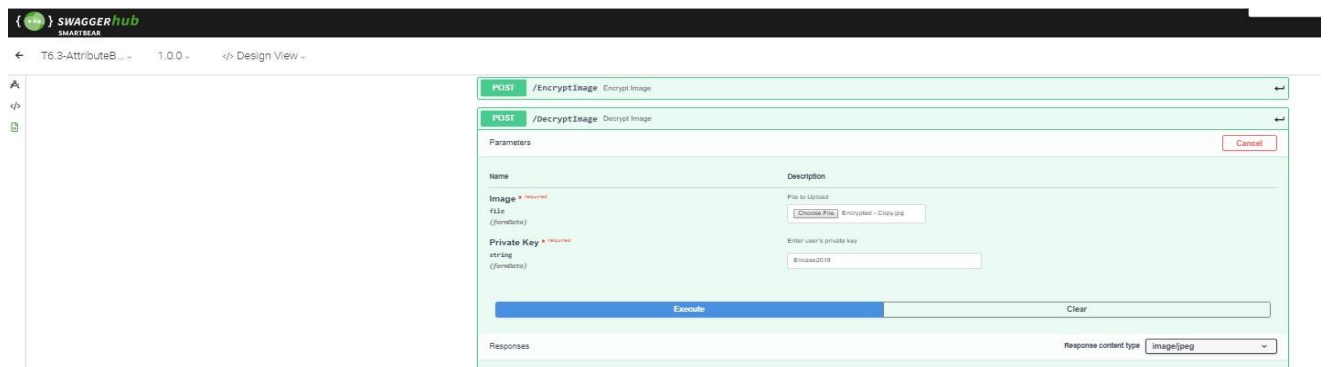


Figure 13 - Decryption API with User's Private Key

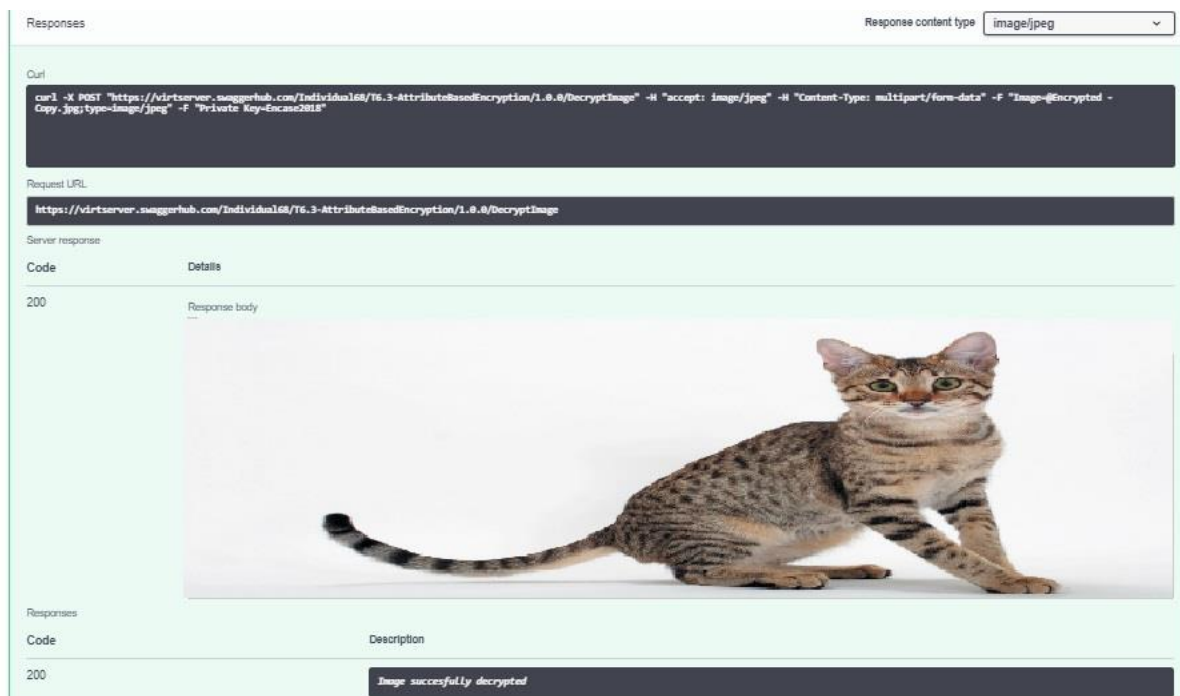


Figure 14 Decryption Response

7. Conclusion

The present document provides the description of the Implementation of browser add-on for content detection and protection of Task 6.4 of the ENCASE project. The sensitive content which is uploaded and shared through OSNs should be protected from malicious activities and users should be able to specify the groups of people that can view their sharing data. The implementation has been focused on the development of image steganography, watermarking and cryptographic techniques as an effort to address this issue. The outcomes of the implementation outlines the importance of these techniques and their feasible integration in an OSN framework for the protection of minors.

8. Copyright and Intellectual Property

The intellectual property will be jointly owned between the Institutions that each of the ENCASE partners. If a project partner decides to move institutions for the duration of the project the Institution to which they move would not become a joint owner, and the ownership will remain with the institution at which partners are originally based.