**Guideline**  
Integration DeepShield Platform: Module B

Guideline how to setup and integrate the technical Secublox platform demonstrator of project DeepShield for Module B.

|  |  |
| --- | --- |
| Document No.: | SECDSMA0012025 |
| Issue: | Secublox GmbH |
| Date: | 04.05.2025 |
| Contacts | langewisch@secublox.com |

|  |  |
| --- | --- |
|  |  |
|  |  |

Revision Record

|  |  |  |  |
| --- | --- | --- | --- |
| Issue | Date | Description / Change | Modified Chapters |
| **SECDSMA0012025** | 04.05.2025 | Final version | All |

# Introduction

This document provides the Guide which makes it able to setup and integrate the technical Secublox platform demonstrator of Project DeepShield for Module B.

## Goals

DeepShield demonstrator for Stage 1 is based on following overall IT architecture divided into four modules:

A diagram of a network

Description automatically generated

The overall mission consists of the following main modules:

* **Module A (create & register content)**: Authentication by integrating cryptographic, digital identity, watermarking together with FPGA and trusted execution environments logic (A.1: Smartphone app / A.2: FPGA-based machine). In additional signed data will be transferred to the decentralized network (Module B).
* **Module B (DLT network):** Prototype of a decentralized network to register digital identities (devices, accounts), image signatures/integrity and image reference content. In additional to verify existing images.

**Module C (verify content):** Prototype of the Artificial Intelligence (AI) services to detect deep fakes of images connected with decentralized network.

* **Module D (administration):** Prototype of the Integration-Framework (Software-Libraries, APIs, Integration) together with the DeepShield-Platform (Dashboard, Backend)

## Module B (DLT)

Module B registers and handle public blockchain registered devices (digital identity of content creators) and accounts (real user, owner of a content creator) together with the deployed DeepShield-Smart-Contracts.

Additionally, it will enable a pre-registered public key of each device, and the original watermarked content can be registered on the public blockchain.

A screenshot of a diagram

Description automatically generated

Figure 1: DeepShield – Stage 1- Module B

# DeepShield Flow

DeepShield flow to demonstrate how to use Blockchain based registered digital identities (devices, accounts) and create authenticated images signed by registered digital identities and watermarked images.

**A diagram of a software development process

AI-generated content may be incorrect.**

Figure 1: DeepShield – Stage 1- Module A/B – Create and register images

Stage 1 focuses on create, embed watermarking and register images in an authenticated way, with focus on data centric security[[1]](#footnote-2). The overview flow is based on following view.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: DeepShield – Stage 1- Module A/B – Create and register images - steps

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Test Data & Code-** **Snippets** | **Comment** | **Blockchain** |
| **0:DeepShield Smart-Contract** | DeepShield Contract issued by Secublox-Account  Secublox-DeepShield-Library:  # Smart-Contract instance of DeepShield  contract\_instance =  initialize\_connection(node\_url, smart\_contract\_address, abi\_file)  # example python script  https://gitlab.com/secublox-platform/iabg\_deepshield/-/blob/main/Module\_B\_D/images.py | Smart-Contract to register, manage & execute transactions | <https://sepolia.arbiscan.io/address/0xBFd555e7BF08e6EEa58Aa2120a0265136d952Ca5> |
| **0:user account creator (user) registered** | Account: iabg\_creator\_1  Wallet (public key):  0xBA4d9B240F88856E70D25AAce76F99878d1cC947  Wallet (private key):  0xf21d1e7f5201c008a87454fba2d80c6016bfdb3da343145e1a8fdf7fd5757d37 | User accounts are able to register and assign devices / apps | <https://sepolia.arbiscan.io/address/0x1dE101999f77fDa9941A9E440210d819ED7F33BC> |
| **0:app account creator (device/app) registered** | Account: iabg\_cc\_app\_1  Wallet (public key):  0x6841E3AEFB55Eb087F923fd2971Bc642b858F79b  Wallet (private key):  0xaba03fd5f886d97052fb26be51847e2f7de3e7d86b34f234ee2ec797e114a113 | app account is able to register images | <https://sepolia.arbiscan.io/address/0x6841E3AEFB55Eb087F923fd2971Bc642b858F79b> |

|  |  |  |  |
| --- | --- | --- | --- |
| Create, watermark and register image (executed inside TEE environment) | | | |
| **1. Create image** | By:  User Account: iabg\_creator\_1  App Account: iabg\_cc\_app\_1  Type: JPEG, e.g. image-RAW | Image created and locally (e.g. file system) stored |  |
| **1.1 Add image** | DeepShield-UI:  <https://deepshield.secublox.com/images>  A screenshot of a computer  AI-generated content may be incorrect. | Add image without encryption and assigned with content creator: iabg\_CC\_app\_1 | Added image raw data available / stored in decentralized network (->UI: image view)  Image raw data: |

|  |  |  |  |
| --- | --- | --- | --- |
| **2. Embade watermark with image** | DeepShield-UI:  A screenshot of a computer  AI-generated content may be incorrect.  Select value for “alpha”  IABG-Implementation:  # Embed the watermark into the image using the selected method  watermarked\_image, ground\_truth\_watermark = watermarking\_method.embed(  image=original\_image,  watermark=watermark,  watermark\_positions=watermark\_positions,  alpha=alpha,  ) | Configure “alpha” to define intensity / visibility of the watermark | Status about watermarking stored in decentralized network (->UI: image view)    Watermarked image: |

|  |  |  |
| --- | --- | --- |
| **3. Encrypt image (optional)** | DeepShield-UI:  A screenshot of a computer  AI-generated content may be incorrect. |  |
|  | Encrypred matermarked image: | |

|  |  |  |
| --- | --- | --- |
| **4 Register image and ownership** | Ownership token of the image  minted for wallet referenced with image content.  Image raw data registered  Data matrix registered | Example :  <https://sepolia.arbiscan.io/tx/0x8f14c1a6eec5c0b23dff9a1ddf909cc07b04b10c1e79860b47843ed047f357d9> |
|  | # example python script  <https://gitlab.com/secublox-platform/iabg_deepshield/-/blob/main/Module_B_D/images.py> | |

# Setup integration scripts

To connect with Blockchain network and register images following activities are needed upfront. This software library (based on Python) connects account(creator, verifier), content creator(like a device or app), content verifier (like ai, deepfake) to the blockchain network in order to initialize, read and write blockchain transactions.

*# update local environment*

sudo apt update

*#* ***Python Version 3.12.3***

*# install python3*

sudo apt install python3-pip

*# install Secublox python package:*

python3 -m pip install --index-url <https://test.pypi.org/simple/> --no-deps package-deepshield-lib==0.0.4

# install web3 package library

pip install web3==6.0.0

# install tools (MacOs)

pip install setuptools

DeepShield-Smart-Contract (e.g.

address: 0x7830f9e05beB24D997b0534590fa71a623B50952

explorer: <https://sepolia.arbiscan.io/address/0xBFd555e7BF08e6EEa58Aa2120a0265136d952Ca5>

) will be updated in parallel while development and to use this library it needs the newest version together with the ABI-file (contract.abi).

Newest contract.abi version will be available at GitLab:

<https://gitlab.com/secublox-platform/iabg_deepshield>

# Integration DeepShield library (Module B)

Following description explains how to connect with the Blockchain-Network and register and handle images with the Blockchain network.

There are following files available to setup and

**README.md:** DeepShield Project Setup Guide

**images.py:** Example test-script how to register, verify and read status of images

**contract.abi:** Interface definition for currently available DeepShield-Smart-Contract (always use the newest version while current development stage), example contract:

**Following function setup the Blockchain connection:**

initialize\_connection(

node\_url,

mct\_contract\_addr,

mct\_file\_abi

) result: contract\_instance

**Example configuration:**

# Connected Blockchain node to read / write transactions

node\_url = <https://arbitrum-sepolia.core.chainstack.com/02233ea7ad5afbf2a952a04e8f5c693f>

# Used DeepShield Smart-Contract and assigned ABI (always use newest version provided by Secublox)

mct\_contract\_addr = 0x7830f9e05beB24D997b0534590fa71a623B50952

mct\_file\_abi = contract.abi

**Following function executes a Blockchain transaction:**

execute\_transaction(

account = administrator public key

key = administrator private key

contract\_instance = from initialize\_connection function

action = This action contains the function(like register\_image) and parameter values

node\_url = <https://arbitrum-sepolia.core.chainstack.com/02233ea7ad5afbf2a952a04e8f5c693f>

)

**Following function registers an image for Blockchain:**

register\_image(

contract\_instance,

public\_key\_content\_creator,

private\_key\_content\_creator,

public\_key\_creator, meta\_data\_link, node\_url) and also mint the token

contract\_instance = getting from this initialize\_connection function

public\_key\_device = creator(account)/device/app public key

private\_key\_device = creator(account)/device/app private key

public\_key\_creator = creator(account)/device/app private key

meta\_data\_link = pinata uri hash

node\_url = [<https://arbitrum-sepolia.core.chainstack.com/02233ea7ad5afbf2a952a04e8f5c693f>](https://arbitrum-sepolia.core.chainstack.com/02233ea7ad5afbf2a952a04e8f5c693f)

)

**Following function gets the status of an image:**

status\_image(

contract\_instance = getting from this initialize\_connection function

mint\_token = mint\_token getting from register\_image function

)

# DeepShield-Platform (Module D)

The dashboard for DeepShield-Platform is available via Secublox cloud services by following URL: <https://deepshield.secublox.com>



Figure 3: DeepShield-Platform Login

Access to the DeepShield-Platform is available by following IABG user-accounts

#### Account Access

The following users have access to the platform:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Account** | **Password** | **Profile** | **Comment** | **Wallet (public key)** |
| iabg\_creator\_1 | @DEEPSCAM2025! | creator | User account with access to DeepShield (Platform, API, Smart-Contract) to manage registered images and assigned content creator app | 0xBA4d9B240F88856E70D25AAce76F99878d1cC947 |
| iabg\_cc\_app\_1 | @DEEPSCAM2025! | App | Raspberry PI / Smartphone (Android) to register authenticated / watermarked images, registered by iabg\_creator\_1 | 0x6841E3AEFB55Eb087F923fd2971Bc642b858F79b |
| iabg\_verifier\_1 | @DEEPSCAM2025! | verifier | Access to DeepShield (Platform, API, Smart-Contract) to verify created images and verification services | 0x72bb13E5d26F4D5EaEDb14E6a2d5CCED62Cd3677 |
| iabg\_check\_1 | @DEEPSCAM2025! | ai | Now access all the images and able to verify the image by verification services | 0x628F07ed04f1a420d7eB4c6223f2EcE685bcFcD7 |

## Images

After login with the account “iabg\_cc\_app\_1” it is possible, to see an overview of all by this device registered images in the Blockchain.

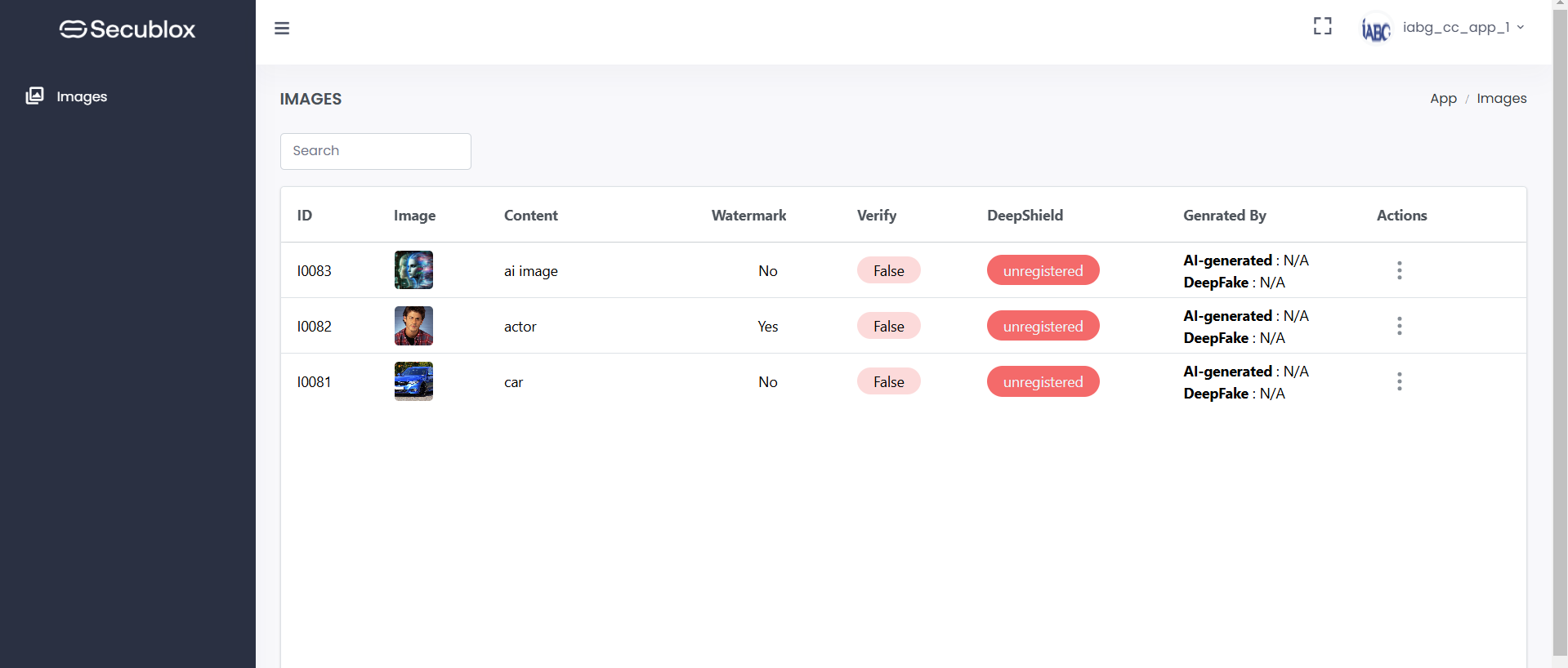


Figure 4: DeepShield – Images table view

After open the info of a single image a popup-window shows the image view together with metadata.

We have 3 types of encryption image

1. **No encryption image** when we open image-info popup.

|  |  |
| --- | --- |
|  |  |
| Figure 5: DeepShield – Images view | Figure 6: DeepShield – Images properties view |

When we click on meta link we just redirected to new window with meta information’s.

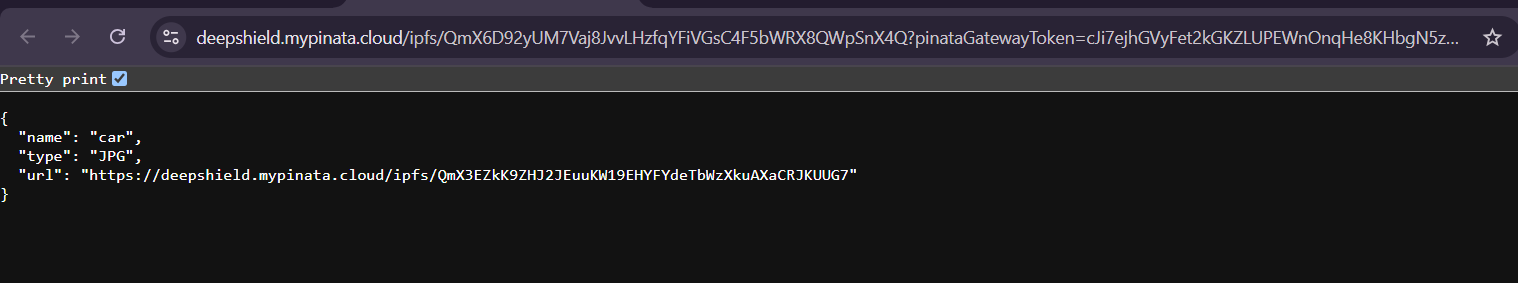


Figure 7: DeepShield – Image meta info

eg. link :- <https://deepshield.mypinata.cloud/ipfs/QmX6D92yUM7Vaj8JvvLHzfqYFiVGsC4F5bWRX8QWpSnX4Q?pinataGatewayToken=cJi7ejhGVyFet2kGKZLUPEWnOnqHe8KHbgN5z8KP3bEoOQEw6nQ-mEsQAELK6AWd>

When we click on **Content Creator (device/app) or Content Creator (Account)** then we just redirected to wallet with all the transaction

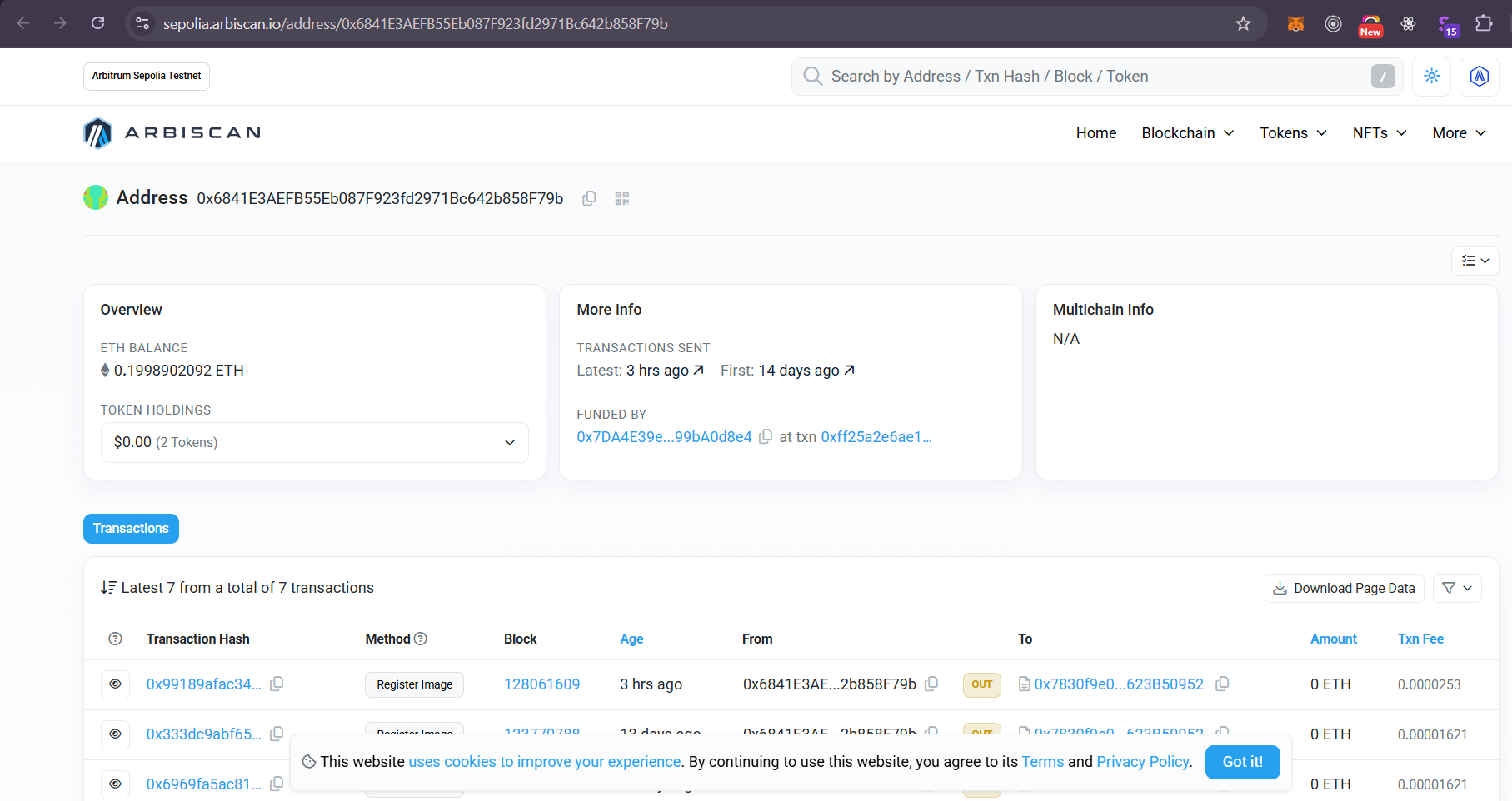
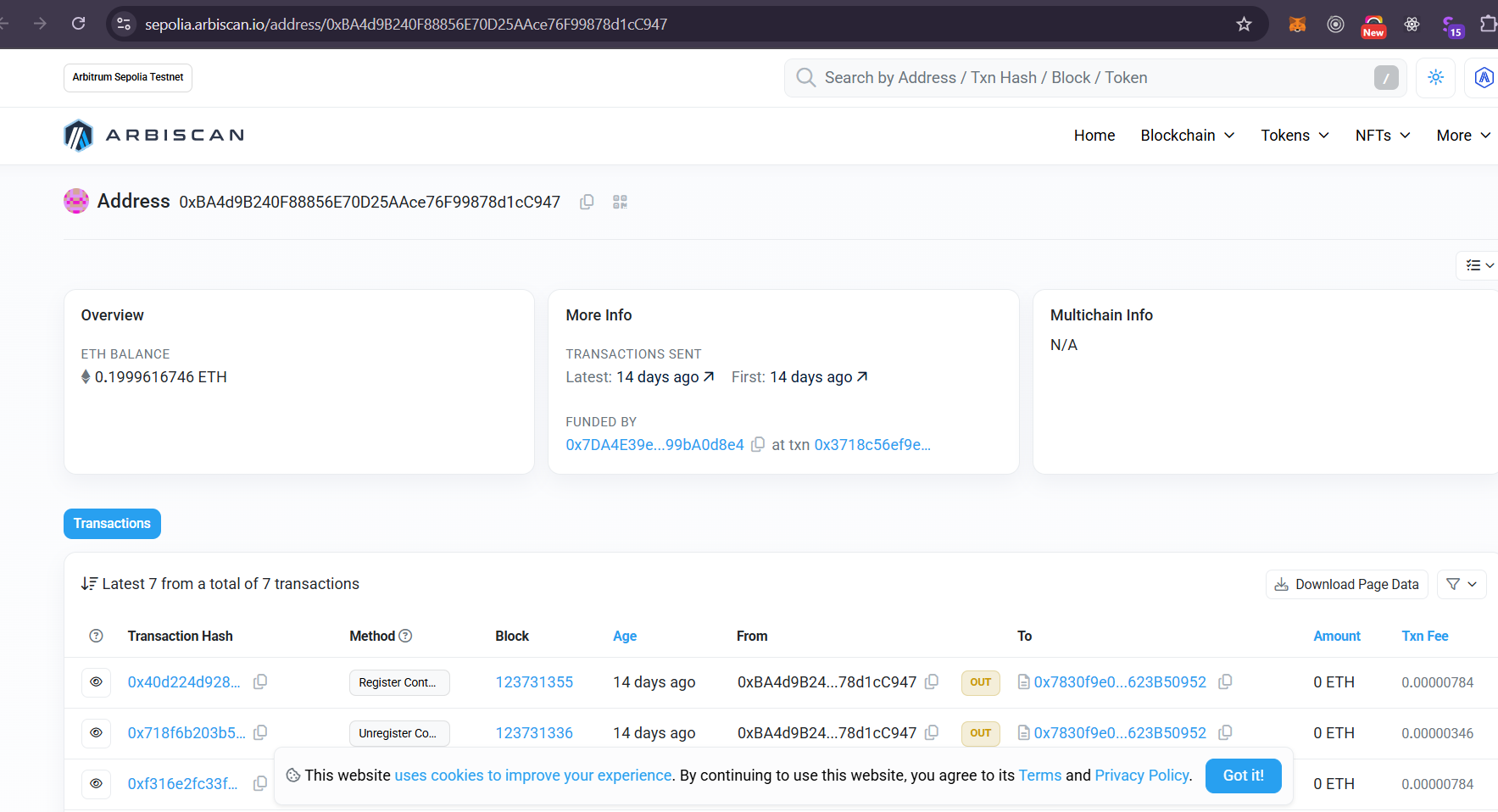
 

Figure 8: DeepShield – Image wallet info

1. **Encryption** **with private key** image when we open image-info popup.

|  |  |
| --- | --- |
|  |  |
| Figure 9: DeepShield – Images view | Figure 10: DeepShield – Images properties view |

When we click on meta link we just redirected to new window with meta information’s.



Figure 11: DeepShield – Image meta info

eg. link :- <https://deepshield.mypinata.cloud/ipfs/QmYn68nhK19hyP1cSWjpkNTep4jQaYQfzyFtfxJ2p9aGXu?pinataGatewayToken=cJi7ejhGVyFet2kGKZLUPEWnOnqHe8KHbgN5z8KP3bEoOQEw6nQ-mEsQAELK6AWd>

When we click on **Content Creator (device/app) or Content Creator (Account)** then we just redirected to wallet with all the transaction

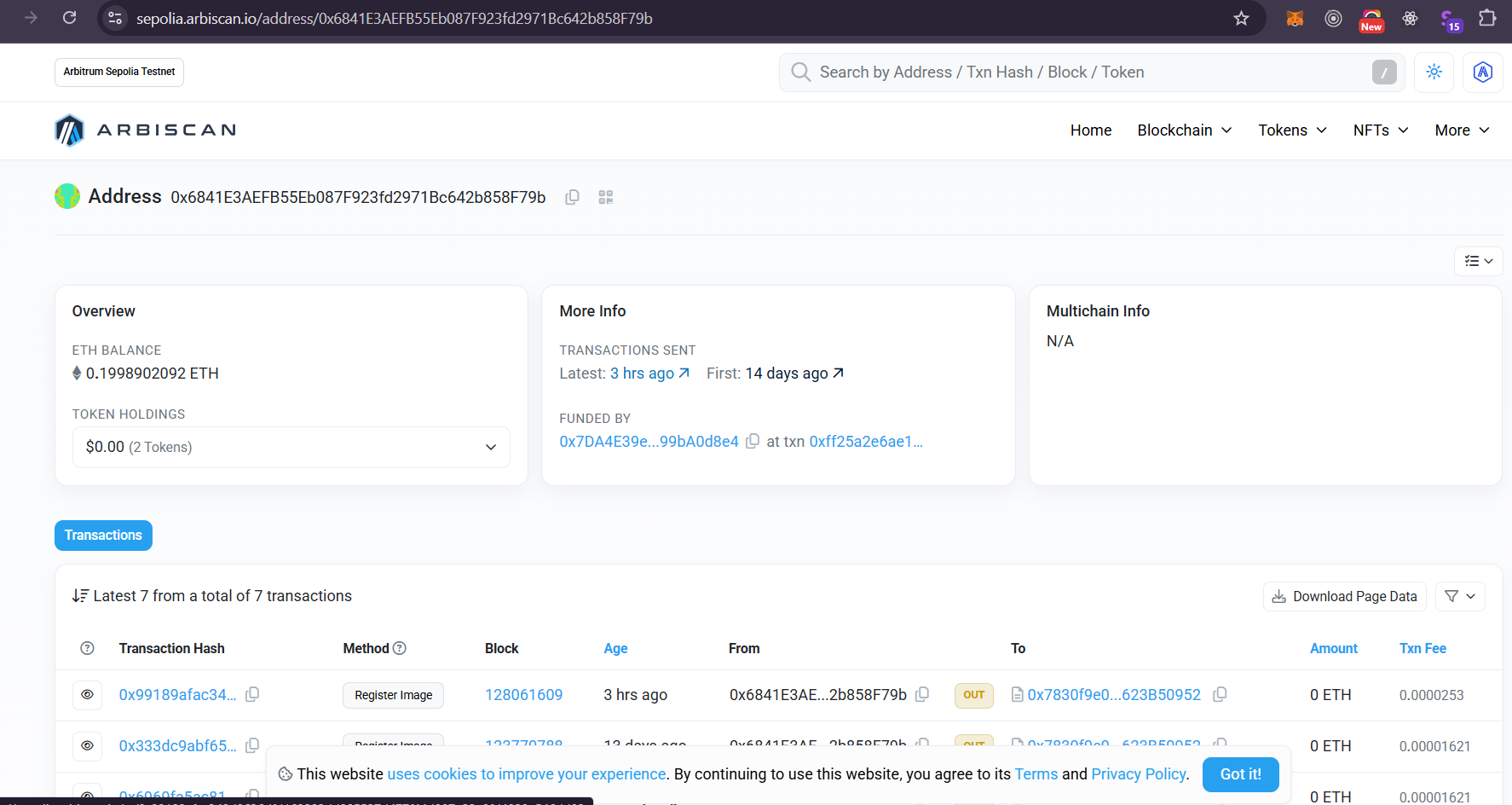
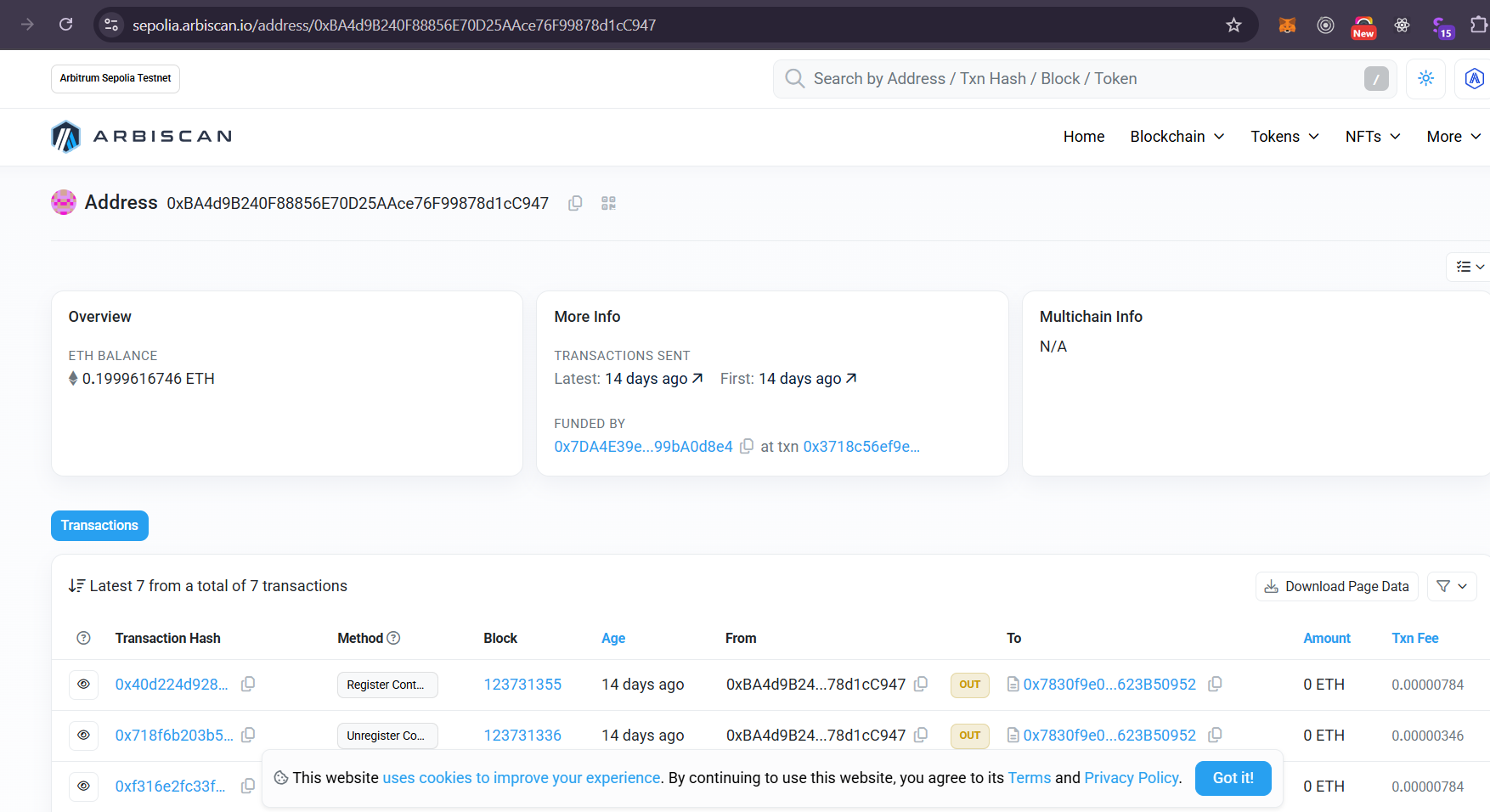
 

Figure 12: DeepShield – Image wallet info

1. **Encryption** **with AES 256 key** image when we open image info popup.

|  |  |
| --- | --- |
|  |  |
| Figure 13: DeepShield – Images view | Figure 14: DeepShield – Images properties view |

When we click on meta link we just redirected to new window with meta information’s.

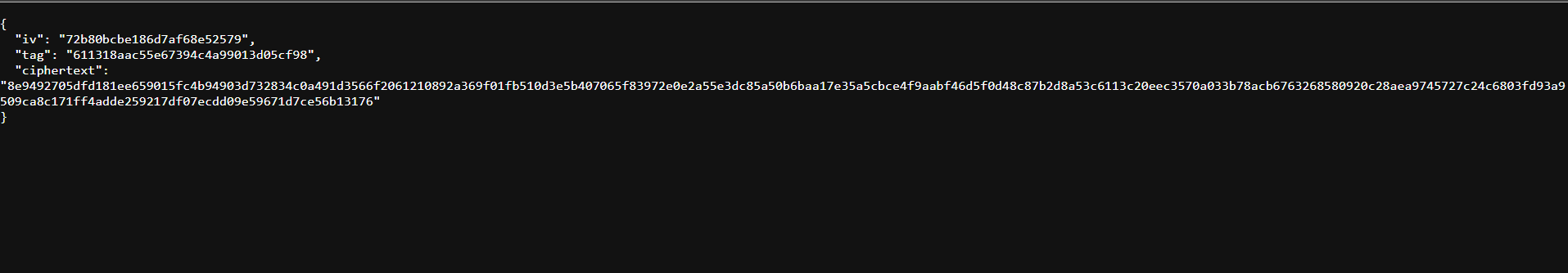


Figure 15: DeepShield – Image meta info

eg. link :- <https://deepshield.mypinata.cloud/ipfs/QmZumGEqkAW8ZN3gcesDk43eyVsBxBtjpgsyaH7yj4xLpY?pinataGatewayToken=cJi7ejhGVyFet2kGKZLUPEWnOnqHe8KHbgN5z8KP3bEoOQEw6nQ-mEsQAELK6AWd>

When we click on **Content Creator (device/app) or Content Creator (Account)** then we just redirected to wallet with all the transaction

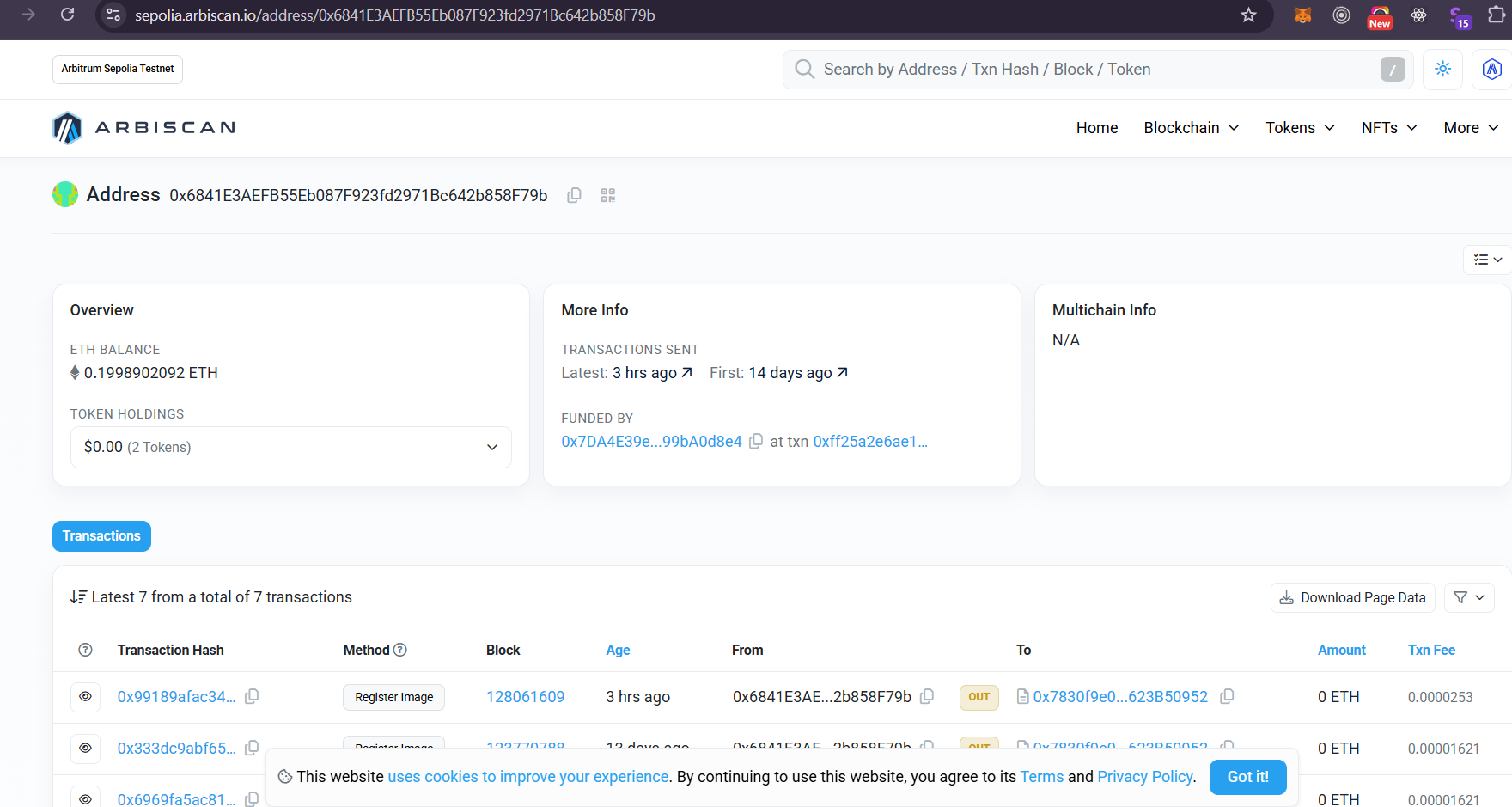
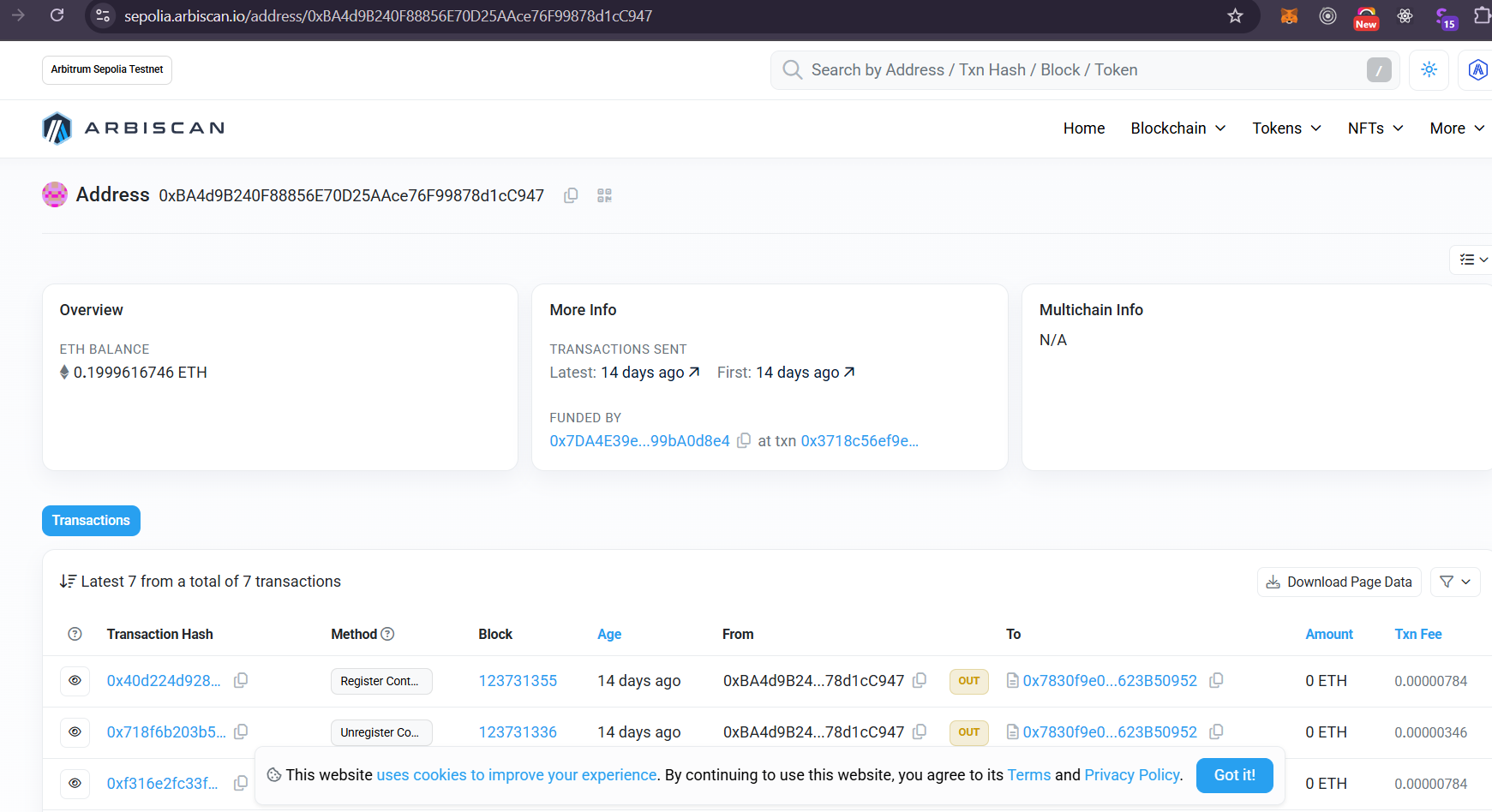
 

Figure 16: DeepShield – Image wallet info

If the image is registered and when we click on register of **DeepShield** column then we just redirected to transaction details.

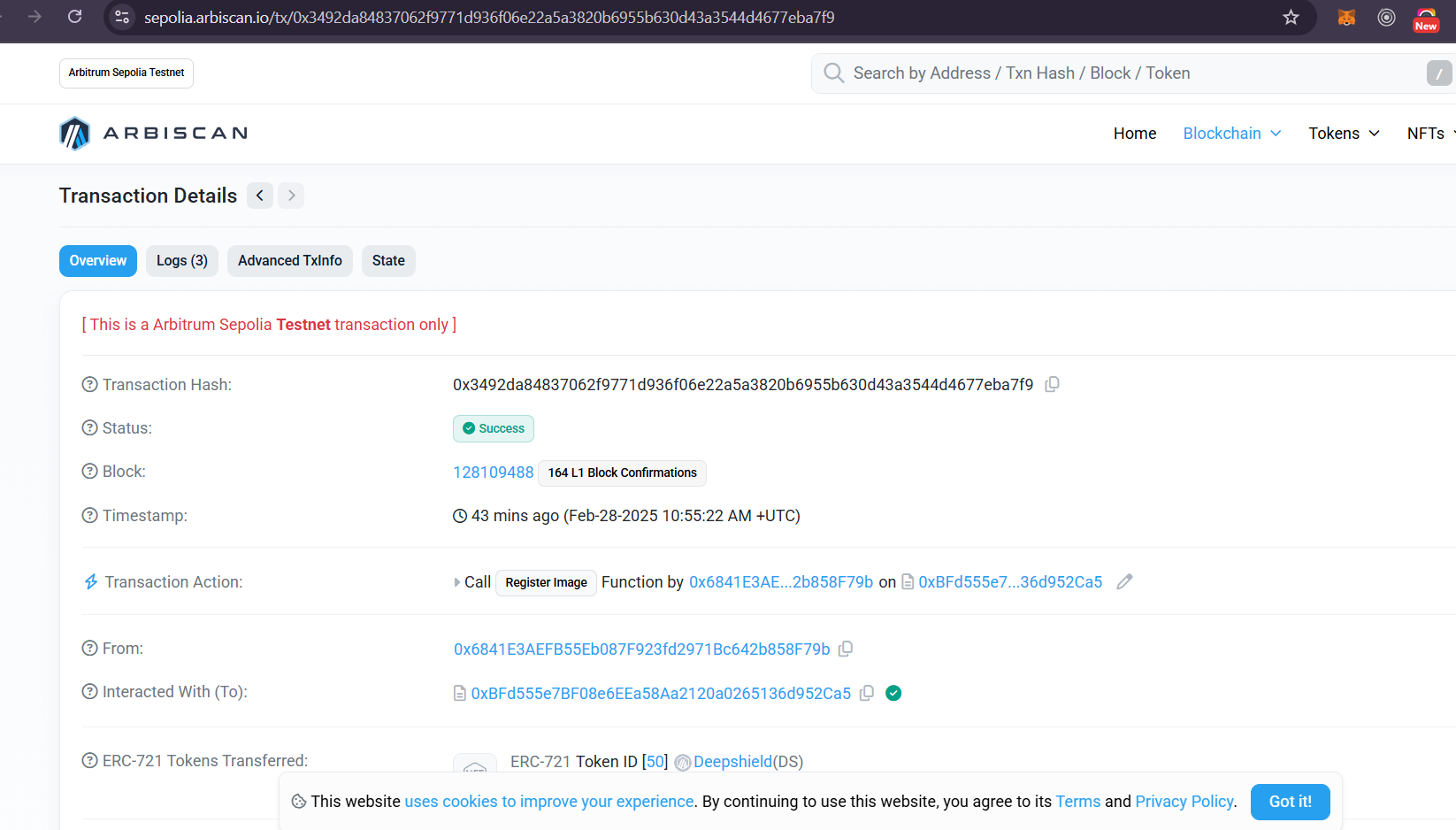


Figure 17: DeepShield – Image wallet info

After login with the account “iabg\_creator\_1” it is possible, to add new image.

1. Encryption Type => No Encryption

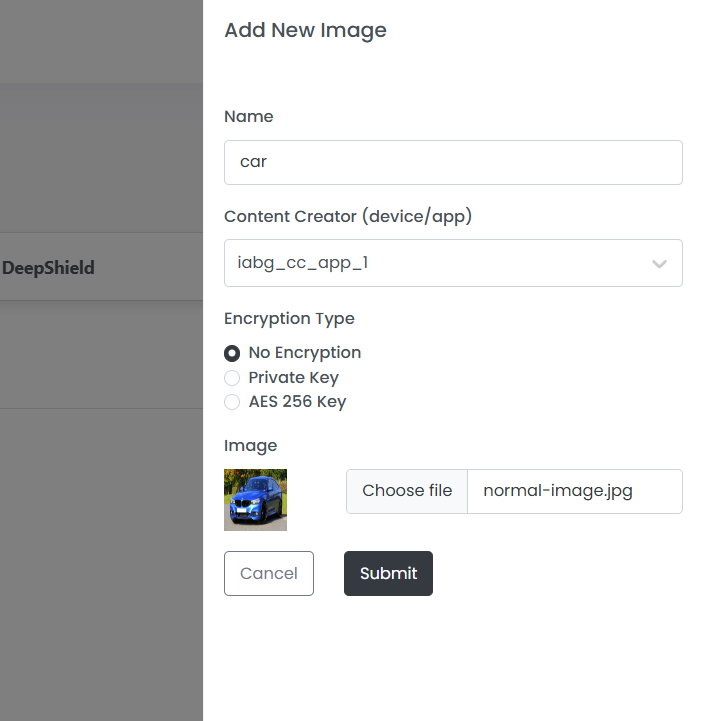


Figure 18: DeepShield – Add new image

1. Encryption Type => Private Key

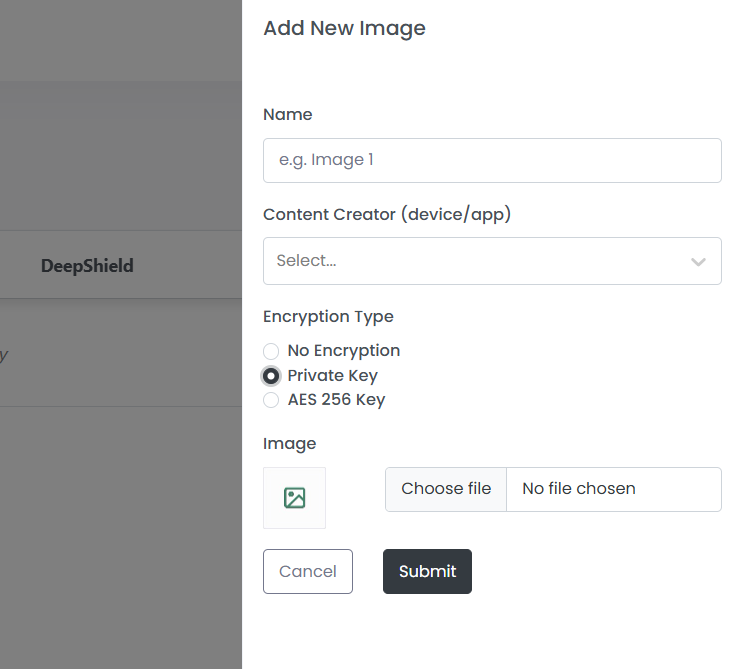


Figure 19: DeepShield – Add new image

1. Encryption Type => AES 256 Key

When we select option of encryption with AES 256 key then we have select AES key from drop-down and make sure to select Content creator (device/app) then only the keys dropdown AES keys will show.

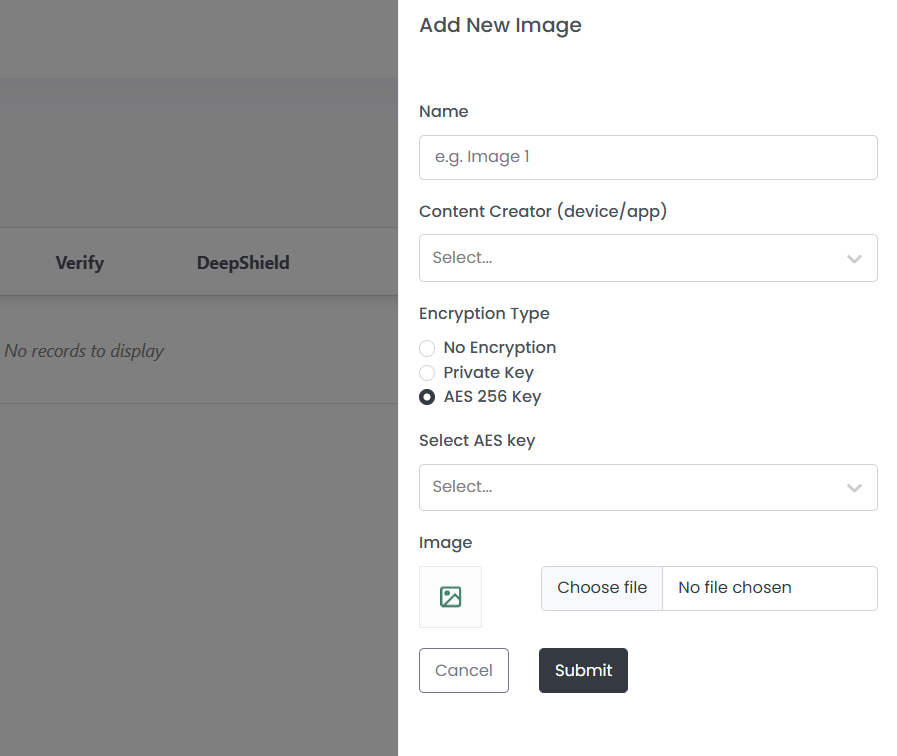
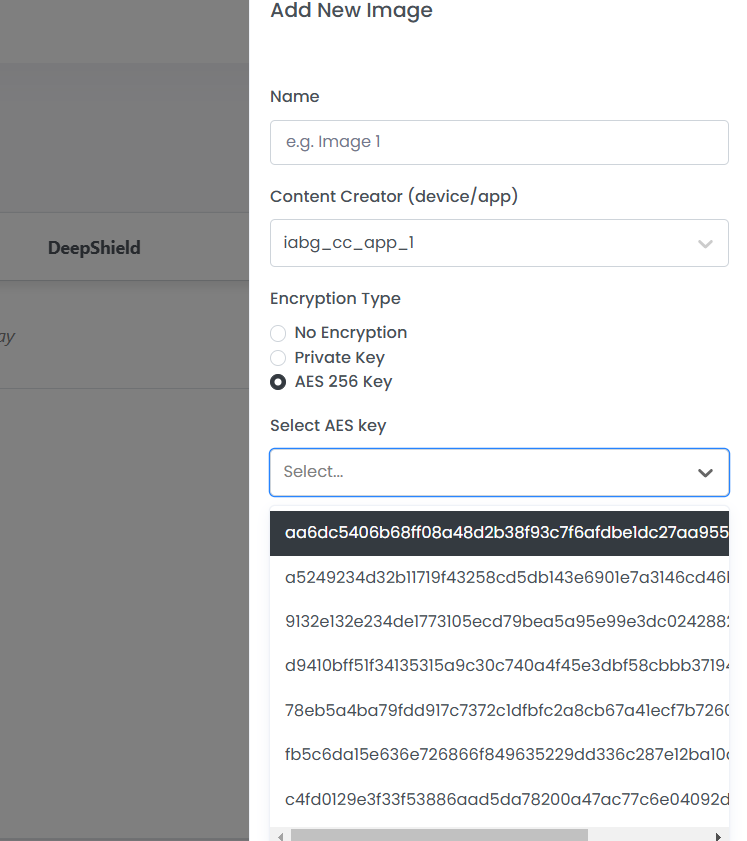
 

Figure 20: DeepShield – Add new image

After login with the account “iabg\_cc\_app\_1” it is possible to register the created image.

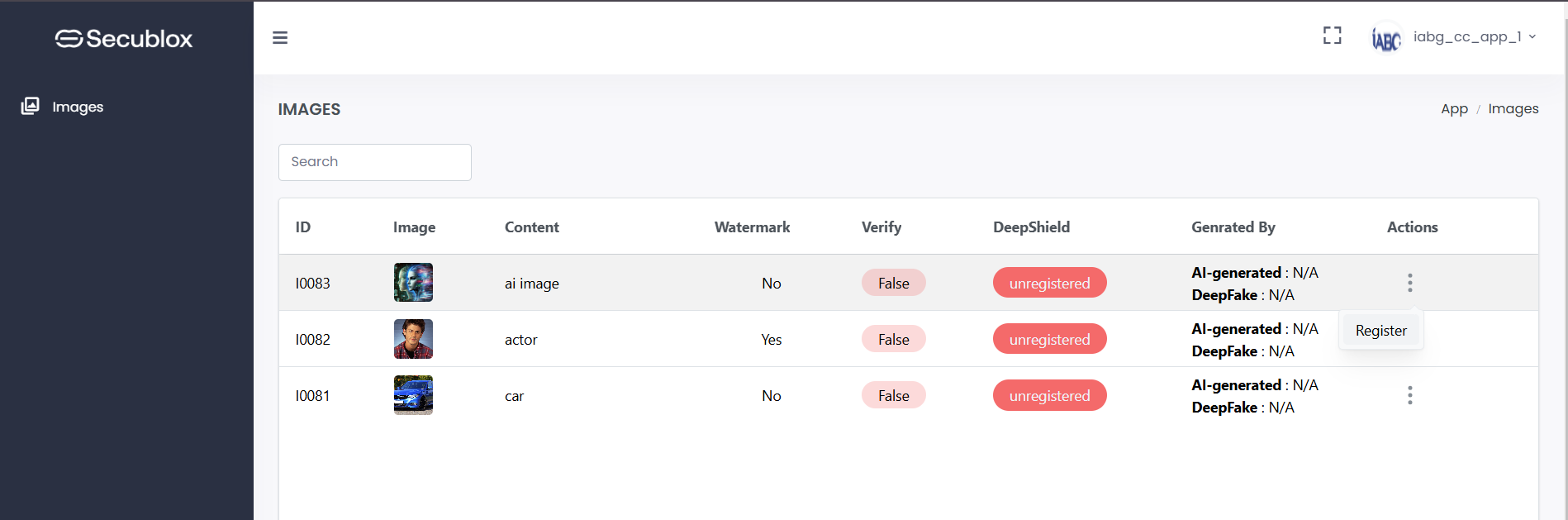


Figure 21: DeepShield – Register image

****

Figure 22: DeepShield – Register image

# Uninstall Libraries

To uninstall the DeepShieldlibrary following console command is needed:

pip uninstall package-deepshield-lib

To check installed version(s) following console command is available:

pip list (package-deepshield-lib Version)

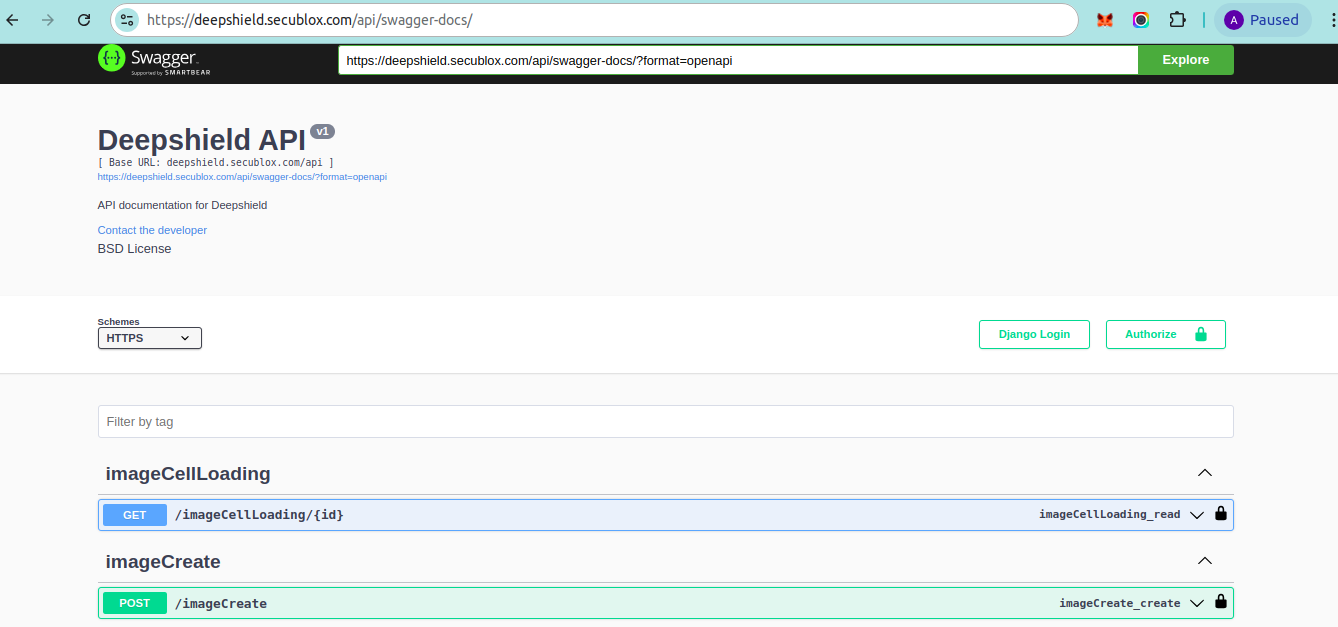
# DeepShield Api Documentation

This document outlines how to access and use the DeepShield API to manage and retrieve image-related data. The API offers endpoints to list images along with associated metadata such as creator details, encryption information, and status indicators.

Document link ->

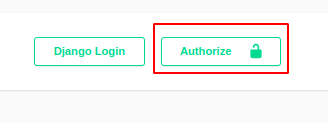
<https://docs.google.com/document/d/14uGKFeEPBSsG1CyujuCSqRtq8S8XAw7Xwmfsb-hGrcc/edit?tab=t.0>

**Step 1**. **Open the Swagger UI**  
Navigate to: <https://deepshield.secublox.com/api/swagger-docs/>

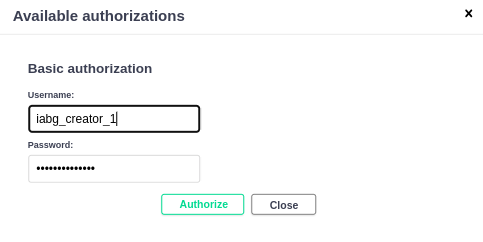


**Step 2. Authenticate**

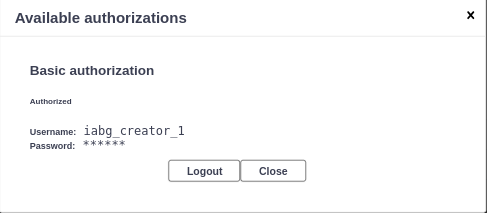
* Click the Authorize button.



* When the login popup appears, enter your credentials.
* Click the Authorize button within the popup.



* Once authenticated, you can view and interact with the API endpoints.

****

## **Endpoints**

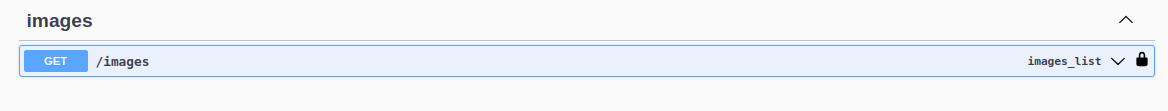
*Retrieve a list of images and their associated details.*

**Endpoint:** GET /images

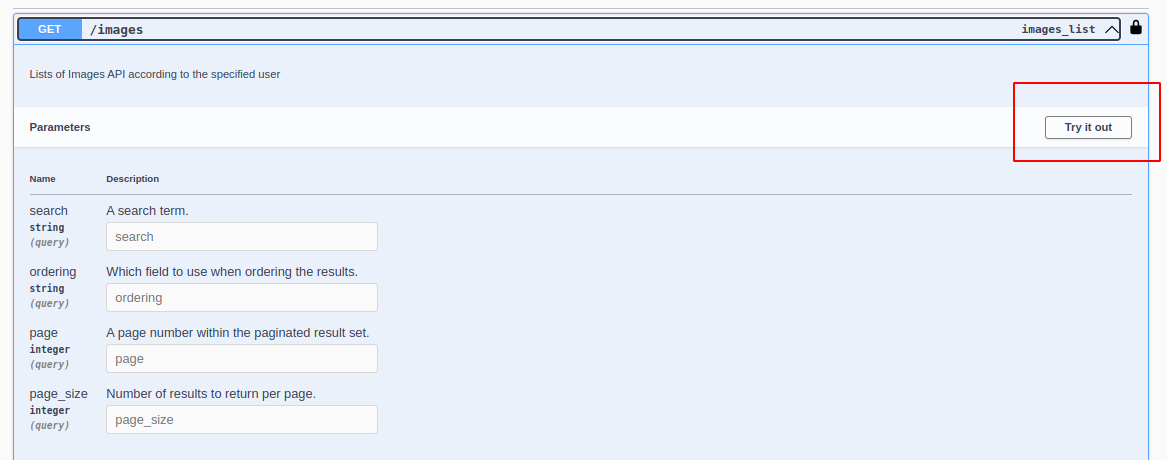
**Request Parameters:**

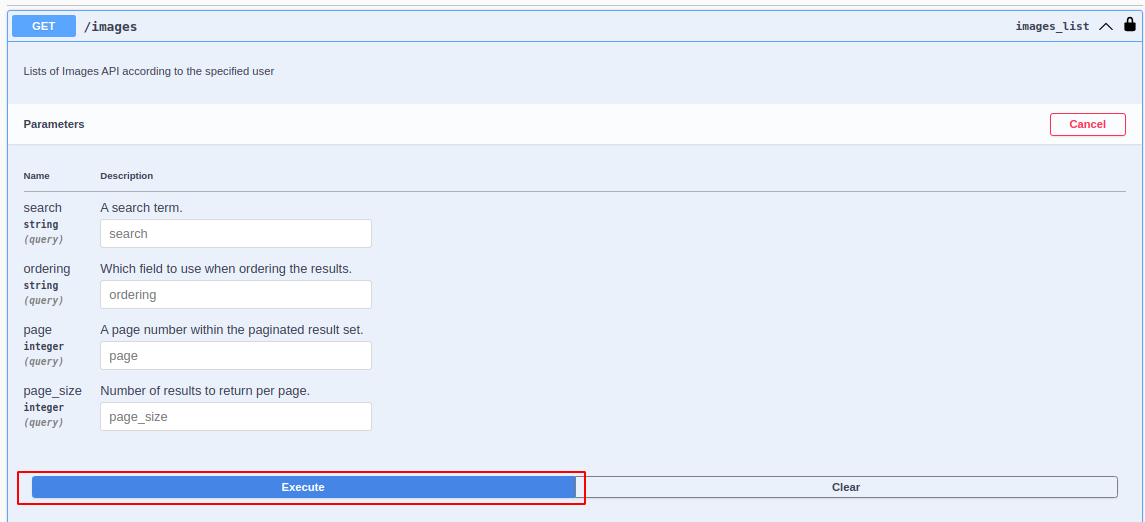
* **search (optional, query)** Filter images by name. The search supports partial matching (e.g., a slightly misspelled image name).
* **page\_size (optional, query)** Specifies the number of records per page.
  + *Default*: 10
  + *Example*: page\_size=20
* **page (optional, query)** Used for pagination when the number of records is large.
  + *Example*: page=1 for the first page.
* **ordering (optional, query)** Sort records by the created\_at field.
  + To sort in ascending order, pass created\_at.
  + To sort in descending order, pass -created\_at.

**Request Example:**

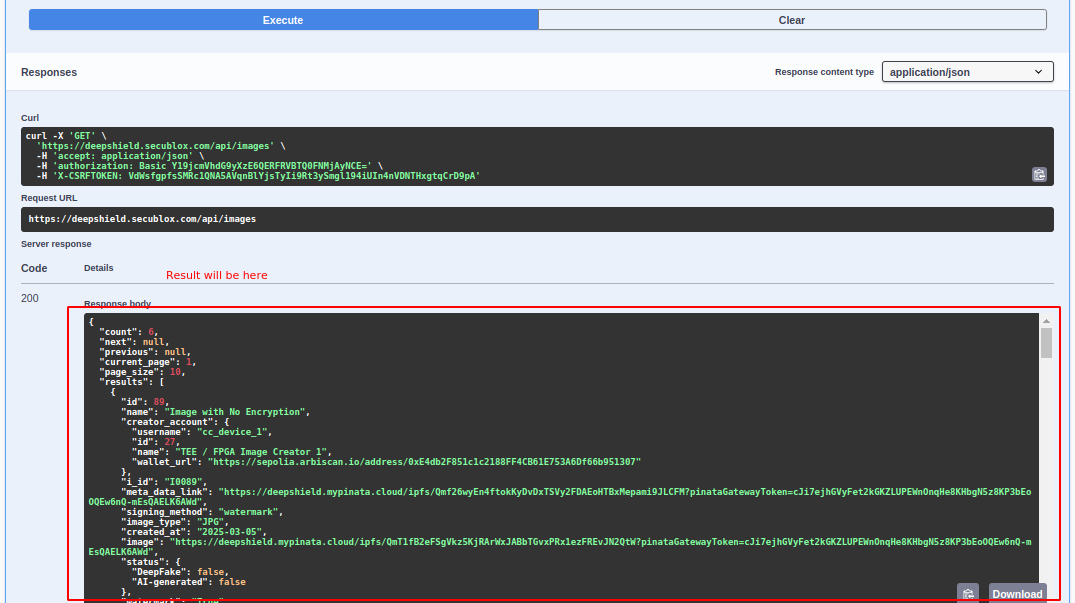
****

Click on down arrow button it look like this

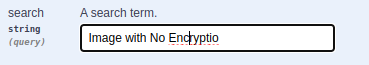


After click on **Try it out** and Click on **Execute** button currently no need to pass **search, ordering, page, page\_size  
  
**

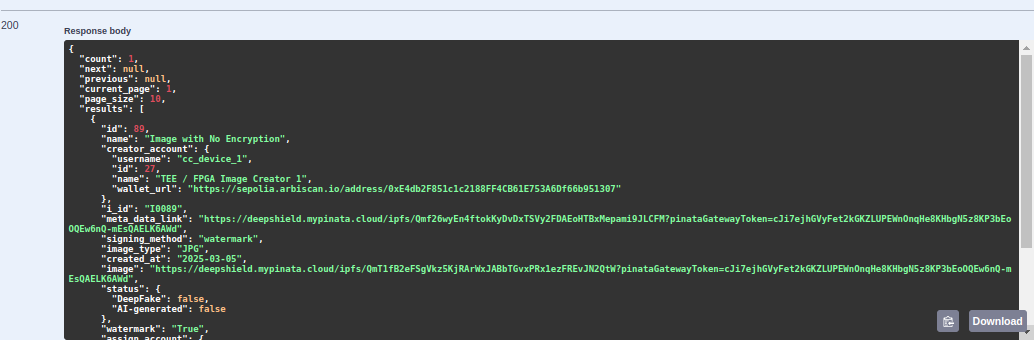
Response will appear here (all image show related to login **creator/app/device** etc)

****

**Search Box**: If you want to search by image name, enter the image name in the search box. I entered an image name with a missing character in the search box and clicked the execute button. The results were displayed based on the given name.



Only one result is displayed based on the search text, and the count shows the total number of records available.

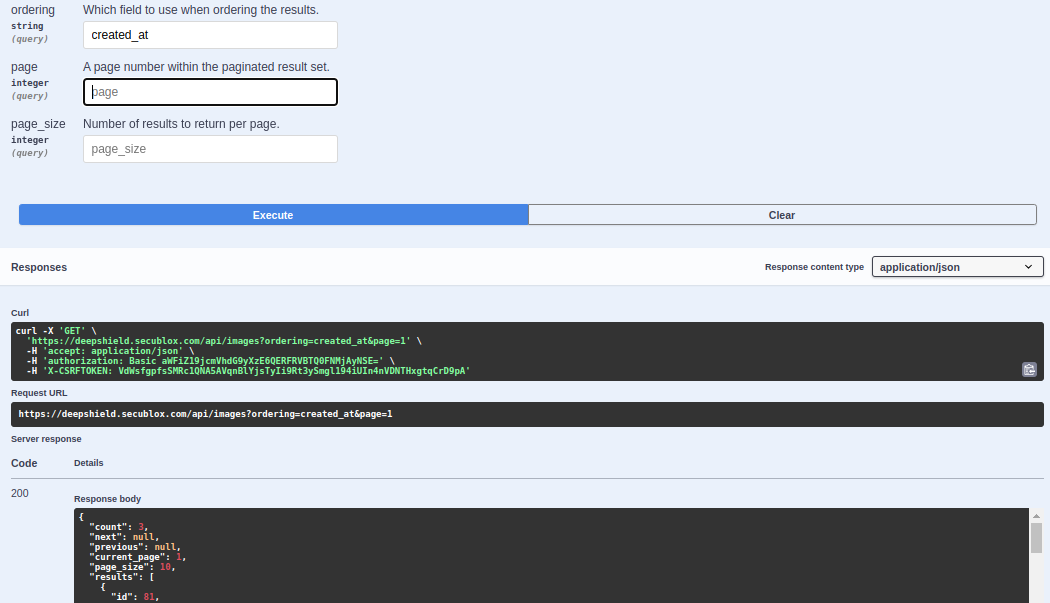


**Page\_size (Optional) :** page\_size determines how many records to display per page. If the page\_size field is set to 20, it will display the last 20 records. By default, it shows 10 records. This is an optional field.

**Page(Optional)**: A page is used when we have a large number of records. Typically, we display 10–20 records per page, dividing the data into multiple pages for better navigation and readability currently we can ignore this becuase we have less data

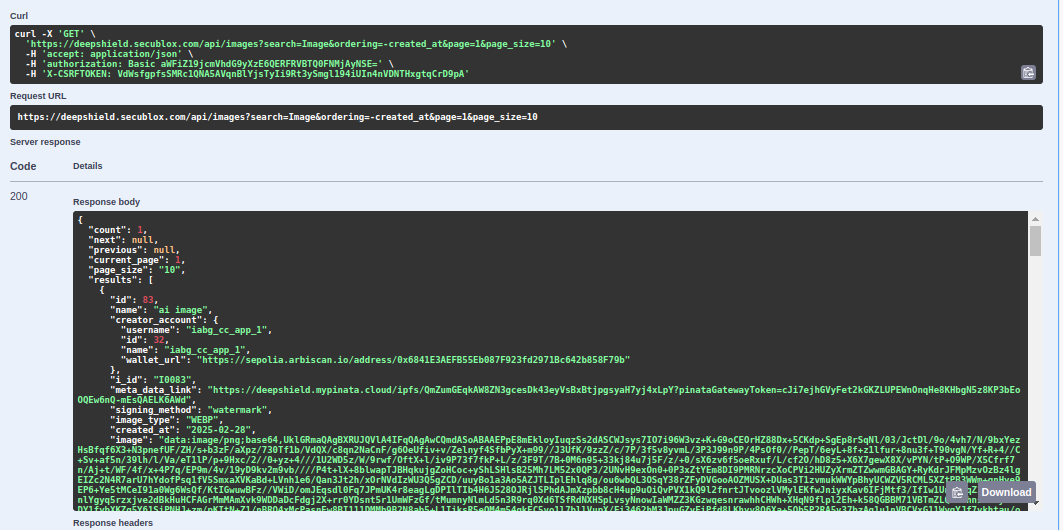
**Ordering(Optional)**: Ordering is used to arrange records in ascending order based on the created\_at field. If we pass -created\_at, it will return the records in descending order.

**Example:**

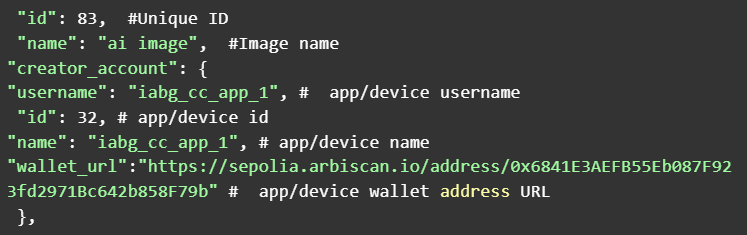


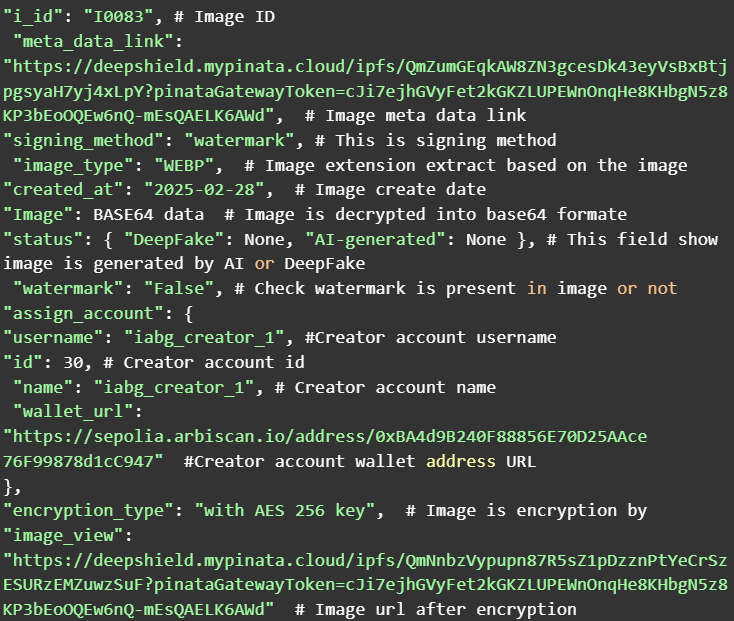


**Examples Response:**



#### **JSON Response:**



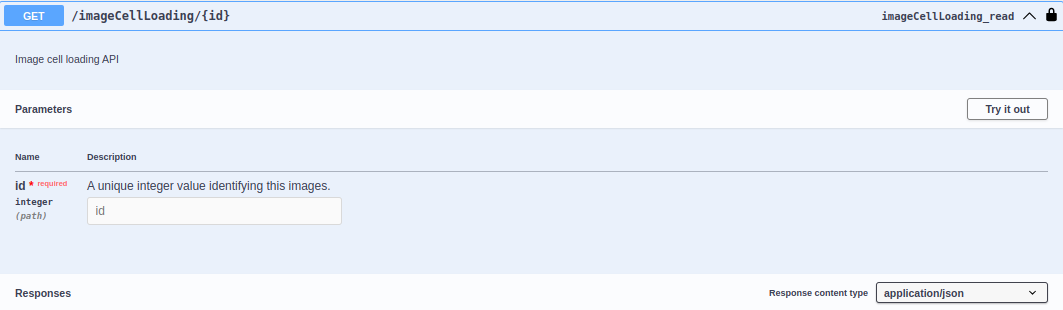


## **Image Status Check and Registration Endpoints**

These endpoints allow you to verify if an image is registered/verified on the blockchain and to register an image by providing the image ID obtained from the /image API.

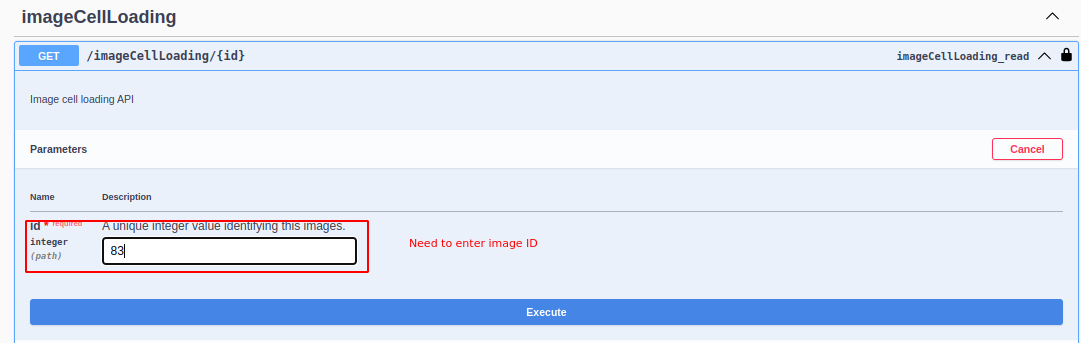
#### **Endpoint:** GET /image/status **Request Parameters:**

* **image\_id** (required, query parameter)  
   The unique identifier of the image as provided by the /image API.

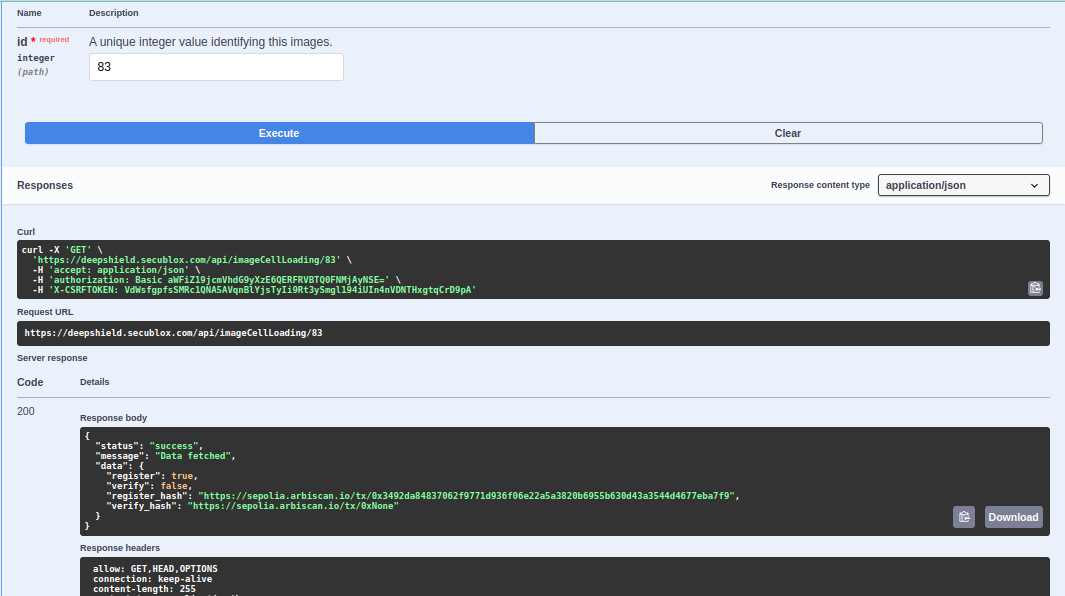
**Example Request:**

After click on **Try it out** button it look like this and enter image id getting from

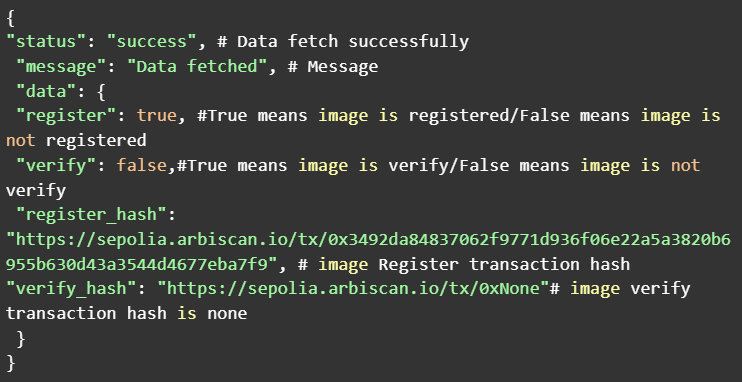
**/image** API this ID need to pass here  
  

click on **Execute** button



#### **JSON Response:**

****

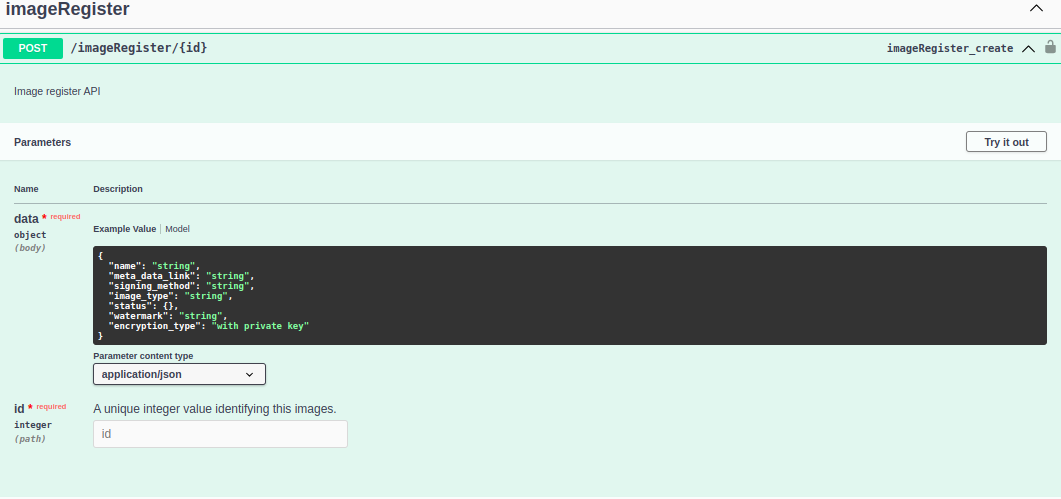
### **Register an Image**

This API registers an image on the blockchain using the image ID. To perform registration, the image ID obtained from the /image API must be passed in the request body.

#### **Endpoint:** POST /image/register **Request Body**:

* register (required, JSON body key)  
   The image ID that you wish to register

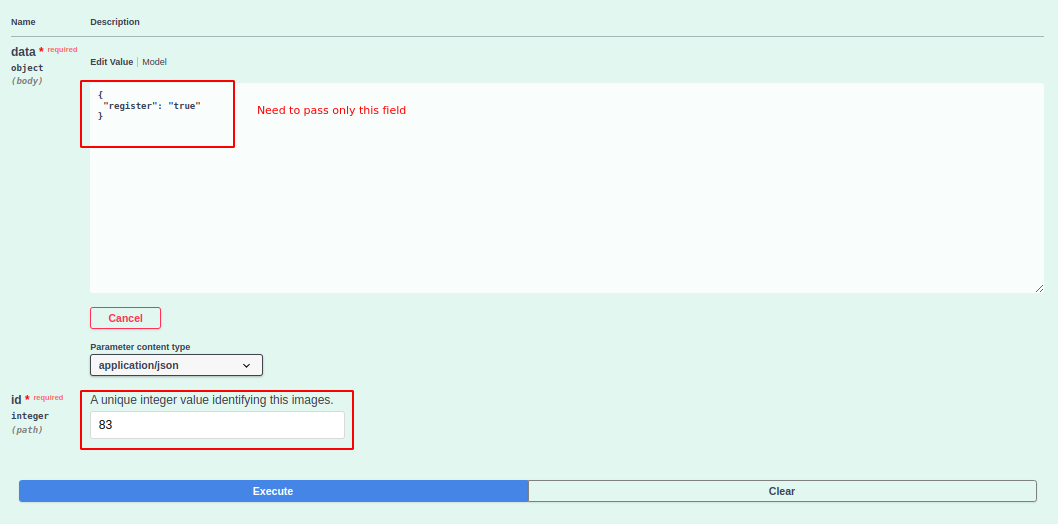
**Example Request:**



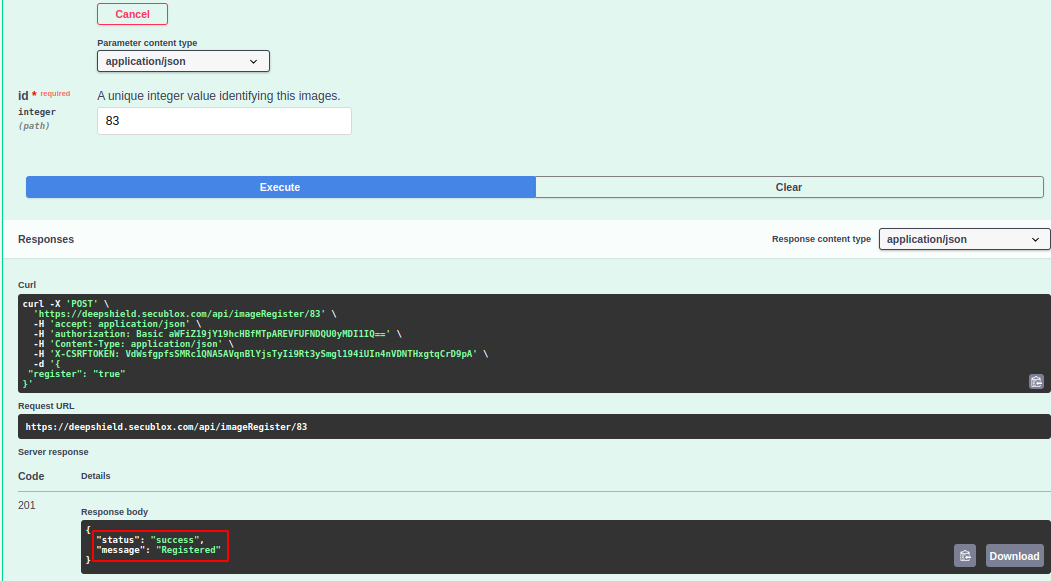
After click on **Try it out** button it look like this and add register key in body and enter image id getting from

**/image** API this ID need to pass here

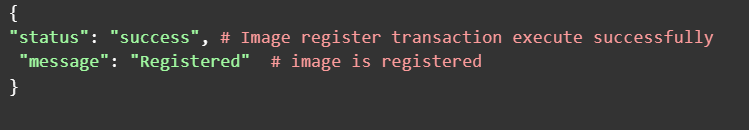




After click on the **Execute** button image will be register



#### JSON Response:



### **Authentication & Additional Notes**

* **Authentication:** Before calling these endpoints, you must authenticate via the Swagger UI by clicking the **Authorize** button and entering your credentials. Once authenticated, you can access and test these endpoints.
* **Integration Tip:** Use the image ID returned from the /image API in both endpoints. For status checks, include it as a query parameter; for registration, include it in the JSON body with the key register.
* **Blockchain Links:** The response for the status check includes direct links (via register\_hash and verify\_hash) to the blockchain explorer (e.g., Sepolia Arbiscan) to verify the transaction details.

1. https://www.paloaltonetworks.com/cyberpedia/data-centric-security [↑](#footnote-ref-2)