**Abstract**

**Module B – Decentralized Network (DLT)**

DeepShield: A distributed and cryptographic approach for authenticating digital content, based on Trusted Execution environments (TEE), establishing ownership, and detecting unwanted manipulations with focus on the Decentralized Network, connected with the overall DeepShield approach.

Stage 1 (Months 1–7, timeboxed, final demonstration 28th Mail 2025):

Research and develop core technologies; conduct initial testing and demonstration.

|  |  |
| --- | --- |
| Document No.: | **B\_DLT\_Abstract** |
| Issue: | A |
| Date: | 12.05.2025 |

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

**Revision Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Issue** | **Date** | **Description / Change** | **Modified Chapters** |
| A | 12.05.2025 | Delivered version | All |

Contact: langewisch@secublox.com

# **Table of contents**

[Table of contents 2](#_Toc197259868)

[Introduction 3](#_Toc197259869)

[Overall Goal DeepShield Approach 3](#_Toc197259870)

[Focus Module D 3](#_Toc197259871)

[List of Abbreviations 4](#_Toc197259872)

[DeepShield Overall Approach 5](#_Toc197259873)

[Implementation A.2 – FPGA VCD 6](#_Toc197259874)

[Requirements: Module D – DeepShield Platform for Management & Integration 7](#_Toc197259875)

[Dashboard 8](#_Toc197259876)

[Accounts 11](#_Toc197259877)

[Content Creator 12](#_Toc197259878)

[Content Verifier 13](#_Toc197259879)

[Images 14](#_Toc197259880)

[Semantic Integrity 15](#_Toc197259881)

[Wallets 15](#_Toc197259882)

[AES-256 16](#_Toc197259883)

[Used Components 17](#_Toc197259884)

[Platform API 19](#_Toc197259885)

[Endpoints 21](#_Toc197259886)

[Image Status Check and Registration Endpoints 27](#_Toc197259887)

[Register an Image 29](#_Toc197259888)

[Authentication & Additional Notes 31](#_Toc197259889)

# Abstract

The rapid development of digital technologies in recent years has opened both fascinating and unsettling possibilities, particularly **in** synthetic content and deepfakes. This sophisticated AI-generated audio, image and video content is now so realistic that it is almost indistinguishable from real footage.

With the exponential increase in the performance of systems and language models based on artificial intelligence (AI) and machine learning (ML), not only the quantity but the quality of deepfakes is improving rapidly. This poses dangers – for trust, security and the perception of reality in our society.

Efforts to combat deepfakes are currently focused on two major areas:

* **Detection:** Development of AI algorithms to identify deepfakes
* **Prevention:** Implementation of authentication mechanisms for digital content.

Despite significant progress in these areas, major challenges remain, such as the generalizability and scalability of AI deepfake detection systems and the establishment of a manipulation-resistant standard for image metadata.

## Overall Goal DeepShield Approach

Development of a prototype that reliably detects deepfakes images from various media content and/or protects existing infrastructures from the use of deepfakes with preventive measures. To address these goals the overall DeepShield approach was divided between different independent modules (A.1, A.2, B, C, D) and will be integrated, to complete the overall DeepShield approach, demonstrated at the end of stage 1.

## Focus Module B

This document focusses on the abstract for the design, concept, implementation and demonstration of module B a prototype demonstration of the Blockchain based Decentralized Network for Registration and Management of all relevant digital objects like user accounts, devices and images along the overall DeepShield approach and interoperable integrated with different platforms.

**Technology:**

Prototype of the Decentralized Network will demonstrate and met following criteria to support the overall DeepShield goals:

* How user accounts, devices and image content can be reliably registered and verified.
* Integration into the overall DeepShield approach
* Resilience in your team and in your solutions.
* Different use cases will show how authenticated and secured image data, user accounts and devices are registered in decentralized Blockchain network by the DeepShield Platform
* Scalability and adaptability to different digital platforms

## List of Abbreviations

Below in Table 1 is a list of abbreviations used in this document.

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| BC | Blockchain |
| DLT | Distributed Ledger Technology |
| DN | Decentralized Network |
| DSC | DeepShield |
| IPFS | InterPlanetary File System |
| KYC | Know Your Customer |
| LLM | Large Language Model |
| NFT | Non-Fungible Token |
| PRNG | Pseudo-Random Number |
| TEE | Trusted Execution Environments |
| TRL | Technology Readiness Level |
|  |  |
|  |  |
|  |  |

## DeepShield Overall Approach

The DeepShield approach will focus on following requirements to demonstrate the defined goals by following functionalities:

|  |  |
| --- | --- |
| **A: Content Creation & Secure Watermarking** |  |
| A user captures a photo or video with on a public blockchain registered device.  These photos or videos are watermarked on-device within a secure, trusted execution environment making use of the private device key and then stored locally or elsewhere. |  |
|  |  |
| **B: Content Registration**  Additionally, to the preregistered public key of the devices, the original watermarked content can be registered on a public blockchain. This allows for ensuring context integrity using further meta information such as reference vector embeddings. |  |
| **C: Content Verification & Context Integrity**  The registered public key enables verification of the content’s authenticity, confirming its origin from a specific device based on the watermark. Degradations in the watermark indicate content manipulation.  Using meta information such as reference vector embeddings, contextual integrity can be verified, in a multilayered approach, helping to prevent disinformation. |  |
| **D: Verification and Revocation via Distributed Ledger**  Public Key Management: Verifiers can retrieve the device’s public key from the blockchain to authenticate the watermark and digital signature, ensuring transparent, verifiable content authenticity.  Revocation Notices: The ledger will contain information about revoked or compromised keys, ensuring that only active and trusted devices can be verified  Smart Contracts for Automation: Implement smart contracts to handle device registration, key updates, and revocation seamlessly |  |

# Implementation A.2 – FPGA VCD

To investigate the described challenges the DeepShield demonstrator for Stage 1 will be based on following overall approach divided into following modules:

A diagram of a blockchain network

AI-generated content may be incorrect.

Figure 1: DeepShield – Stage 1 – Main Modules

The overall DeepShield approach 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 detec deep fakes of images connected with decentralized network.

* **Module D (Management & Integration):** Develop a prototype of the Integration-Framework (Dashboard, APIs, Integration)

This document focuses on Module D – DeepShield Platform for Management & Integration.

## Requirements: Module B – Decentralized Network (DLT)

Module B will enable and handle public blockchain registered devices (digital identity of content creators) and accounts (real user, owner of a content creator). 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 2: DeepShield – Module B –Decentralized Network

**Delivery focus for Stage 1 of Module B:**

The delivery of design, concept, implementation, validation and demonstration focuses on following aspects for module B:

**Configure Blockchain-Network:**

* Wallets (accounts / devices)
* Nodes
* Smart-Contract (DeepShieldContract): Create / Deploy Smart-Contracts
* Image tokens: NFTs

**DeepShield Smart-Contract:**

* Register account wallet (public / private key) for digital Identities (person)
* Register device wallet (public / private key) for digital Identities (device, content creator)
* Register image (together with NFT, image reference data and image metadata)
* Register digital watermark of an image
* Sign Blockchain transactions (read / write transactions)
* Blockchain Network connector (device <-> Blockchain)
  + Python-Library: DeepShield-Library

**Create IPFS network (image reference and metadata)**

### DeepShield-Library

A software library (based on Python[[1]](#footnote-2)) will be implemented to connect the content creator (device or mobile app) with the blockchain network, to read and write blockchain transactions (especially DSC).

A diagram of a smart brand

Description automatically generated

Figure 4: Integration by Blockchain library

Key features of the blockchain library to exchange (read/write) DLT-transactions between module A.1, A.2, B, C and D are:

**Module A.1/2**

* Initialize connection with DLT for module A.1/2 with module B
* Register Image (Blockchain, NFT, IPFS)
* Read image information (status, content, metadata) from DLT

**Module C**

* Initialize connection with DLT for module C with module B
* Receive registered image information (Blockchain, NFT, IPFS)
* Verify registered image information

**Module D**

* Initialize connection with DLT for module D with module B
* Administration of the wallets (private / public key)
* Register digital identity of a device
* Register digital identity of an account
* Read, create and write blockchain transactions (DSC)
* Receive and send information for decentralized network

### Blockchain

To support the defined requirements and demonstration of DeepShield: Stages 1, the underlying blockchain for mapping the decentralized network will first be evaluated based existing public Blockchain technologies and protocols, which are based on following components:

**Blockchain: Public Blockchain**

* Arbitrum Sepolia Testnet[[2]](#footnote-3)
* Smart-Contracts:
  + DeepShield-Contract
  + <https://sepolia.arbiscan.io/token/0xbfd555e7bf08e6eea58aa2120a0265136d952ca5#code>
* Transactions:
  + <https://sepolia.arbiscan.io/token/0xbfd555e7bf08e6eea58aa2120a0265136d952ca5>
* Wallets for User-Accounts or Devices:
  + <https://sepolia.arbiscan.io/token/0xbfd555e7bf08e6eea58aa2120a0265136d952ca5#balances>
* Decentralized network based on publicly available nodes:
  + https://arbitrum-sepolia.core.chainstack.com/02233ea7ad5afbf2a952a04e8f5c693f

## Dashboard

The dashboard is accessed via cloud service by following URL: <https://deepshield.secublox.com>

Ein Bild, das Text, Screenshot, Webseite, Website enthält.

Automatisch generierte Beschreibung

Figure 8: DeepShield Platform – Access

The admin access to manage the backend is accessed by following URL:

<https://api-deepshield.secublox.com/admin>

<https://deepshield.secublox.com/admin>

The REST-API of the DeepShield backend is accessed by following URL:

<https://api-deepshield.secublox.com/api/auth/login>

The platform is a multi-tenant system and supports the following user profiles (accounts):

* Administrator: Administrator of the platform. Write and read access to all relevant information for the deployment and operation of the system.
* Accounts: User of DeepShield to manage and execute the use-case (create / verify images) and the assigned devices and wallets.
* Devices: Access to each individual device for configuration and status.

#### Account Access

The following users accounts have access to the platform:

|  |  |  |  |
| --- | --- | --- | --- |
| **Account** | **Password** | **Profil** | **Comment** |
| admin | @DEEPS2024! | superadmin | Main technical administrator of secublox to manage backend (DJANGO) |
| secublox | @DEEPSSECU2024! | administrator | Access to DeepShield Contract and manages all relevant information like registered accounts, devices, smart-contracts, nodes, images |
| c\_creator\_1 | @DEEPSCAM2024! | creator | Access to DeepShield (Platform, API, Smart-Contract) to manage created images and assigned content creator(s) |
| cc\_device\_1 | @DEEPSCCCD2024! | device | Visual Capture Device (FPGA, TEE) to create authenticated image content |
| c\_creator\_2 | @DEEPSSMA2024! | creator | Access to DeepShield (Platform, API, Smart-Contract) to manage created images and mobile\_app account information |
| cc\_app\_1 | @DEEPSCCCA2024! | app | Smartphone (Android) to create authenticated image content |
| c\_verifier\_1 | @DEEPSVER2024! | verifier | Access to DeepShield (Platform, API, Smart-Contract) to verify created images and verification services |
| wi\_check\_1 | @DEEPSWIC2024! | AI-service | Watermark Integrity Check |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

#### Account Settings

Under “Account Settings” profile information of the logged in user is displayed and can be edited. It is also possible to change this user's login password.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Account Profile

In addition, under “Account Settings” the logged in administrator can view and configure the DeepShied-Smart-Contract to register and manage the Blockchain protocol to authenticate or verify images.

#### Blockchain Transactions

Under Profile, the logged in account also has the option to configure the active DeepShield Contract and accessing the transactions carried out under “Wallet” and “Smart Contract”.

|  |  |
| --- | --- |
| A screenshot of a computer  AI-generated content may be incorrect.  Figure 10 Transactions “Smart-Contract” | **A screenshot of a computer  AI-generated content may be incorrect.**  Figure 11 Transactions „Wallet“ |

### Accounts

All users accounts (digital identities of a person) and their rights with access to the platform (+ DeepShield-Smart-Contract and assigned wallets) are managed under the main menu item “Accounts”.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12: DeepShield-Platform - Accounts

For demonstration how deepfakes images can be reliably authenticated and detected by the prototype of DeepShield - Stage 1 following account types are registered:

* Creator: Account which is creator of an image, together with a registered device or mobile app (authentication)
* Verifier: Account which is verifier of an existing image (detection)

Each new account will automatically be assigned with a wallet, to which the digital identity for the Blockchain is identified and to write, read blockchain transactions.

Access through following account profiles:

* Administrator: Administrator access to manage all accounts, content creator / verifier, images, wallets and key tables
* Account:
  + Access to assigned wallets, content creator/verifier, images
  + Created by the administrator
* Content Creator:
  + Access to assigned device (Visual Capture Device) or mobile app (Smartphone - Android) which is registered to create authenticated images
* Content Verifier:
  + Access to AI Service to verify (watermark integrity check) exiting images

**Implemented Features**

The following functions are implemented under the “Accounts” main menu:

* Create new account
  + Assign account to wallet and user profile
* Table view of Accounts
* Edit
* Delete

**Data fields**

The following fields (example) are implemented for the Account object:

|  |  |
| --- | --- |
| * Name * Profile * Username * Password * Email Address * Description |  |

### 

### Content Creator

A Content Creator represents a registered device or mobile app, which creates authenticated images. Under “Content Creator” an administrator can create and manage content creators.

After a content creator like “Visual Capture Device” is registered for the DeepShield-Smart-Contract it has access to the supported functionality of the DeepShield-Smart-Contract.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: DeepShield-Platform – Content Creator

The following functions are implemented under the “Content Creator” main menu:

* Add “Content Creator”
  + Assign to an Account
* Table view of content creators
* Edit
* Delete

**Data fields**

The following fields (example) are implemented for the Content Creator object:

|  |  |
| --- | --- |
| * ID (generated, format: CCxxxx) * Name * Type * Assign Account * Status (registered for DeepShield-Smart-Contract: yes/no) |  |

### Content Verifier

A Content Verifier represents a registered AI Service, which verifies authenticated images. Under “Content Verifier” an administrator can create and manage content verifiers.

After a content verifier like “DeepFake detection” is registered for the DeepShield-Smart-Contract, this service is allowed to verify images.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: DeepShield-Platform – Content Verifier

The following functions are implemented under the “Content Verifier” main menu:

* Add “Content Verifier”
  + Assign to an Account
* Table view of content verifiers
* Edit
* Delete

**Data fields**

The following fields (example) are implemented for the Device object:

|  |  |
| --- | --- |
| * ID (generated, format: CVxxxx) * Name * Type * Assign Account * Status (registered for DeepShield-Smart-Contract: yes/no) |  |

### Images

Registered Content Creators can create authenticated images. Such images are watermarked on-device (on electronic or software level) within a secure, trusted execution environment making use of the private device key and then stored locally or elsewhere.

Afterwards a user can captures / create an authenticated image registered for a public blockchain by a device (e.g. “Visual Capture Device”).

After registration of such an image, it is visible in by the DeepShield-Platform under “Images”.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 15: DeepShield-Platform - Images

The following functions are implemented under the “Images” main menu:

* Table view of authenticated images
* View detailed information of an authenticated image (image raw data, metadata, watermark signature, …)
* Delete image

**Data fields**

The following fields (example) are implemented for the Image object:

|  |  |
| --- | --- |
| * ID (generated, format: Ixxxx) * Content * Date * Metadata * Type * Assigned Content Creator (device or app) * Signing method * Assigned Content Creator (account) * Status (registered for DeepShield-Smart-Contract: yes/no) |  |

### Semantic Integrity

Authenticated images can be semantic analyzed and integrity checked by this functionality. Such images are registered images will be provided for an external integrity check service which executes the underlying semantic integrity logic and provides the results back to the DeepShield platform.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 15: DeepShield-Platform – Semantic Integrity

### Wallets

Each wallet (public blockchain registered) is assigned to an account, content creator or content verifier. A wallet is the digital identity of a real object (account, device, app).

Each transaction is signed (executed) with the private key of such a wallet. Wallets created by the DeepShield-Platform stores public and private keys (which could be also provided and stored in a decentralized way).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 16: DeepShield-Platform - Wallets

Access through the following user profiles:

* administrator: Access to all wallets
* accounts: Access to wallets assigned to logged in wallet (own account wallet, content creator / verifier

The following functions are implemented under the Wallets main menu:

* Table view of wallets
* View detailed information of a single wallet

**Data fields**

The following fields (example) are implemented for the Wallet object:

|  |  |
| --- | --- |
| * ID (generated, format: Ixxxx) * Content * Date * Metadata * Type * Assigned Content Creator (device or app) * Signing method * Assigned Content Creator (account) * Status (registered for DeepShield-Smart-Contract: yes/no) |  |

### AES-256

AES-256 keys are provided for the FPGA Visual Capture Device to encrypt authenticated image content while creation of such content and that this content is transferred to the DeepShield library in a secured way for registration with die Blockchain network.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 16: DeepShield-Platform – AES-256 key tables

## Used Components

The overall software system architecture combines of following components by using following (open source) software libraries:

|  |  |  |
| --- | --- | --- |
| **Used Component** | **Comment** | **License Title / Reference** |
| <https://www.docker.com> | Container virtualization   * Docker Engine 27.2.1 * Docker Compose is 2.31 | Docker Engine is licensed under the Apache License, Version 2.0  [Reference](https://github.com/moby/moby/blob/master/LICENSE) |
| <https://react.dev> | Implementation of user interface.   * React version - 18.2.0 * Node version - v20.17.0 * npm version - 10.8.2 * yarn version - 1.22.22 | Licensed under the Creative Commons Attribution 4.0 International  [Reference](https://github.com/reactjs/react.dev/blob/main/LICENSE-DOCS.md?utm_source=chatgpt.com) |
| <https://www.djangoproject.com> | Web framework (including backend)   * Django version - 3.2.21 | Contributor License Agreement (CLA)  [Reference](https://www.djangoproject.com/foundation/cla/faq/) |
| <https://www.python.org> | Python programming language used for backend and device library   * Python version - 3.9.0 | Licensed under Python Software Foundation License (PSFL) compatible with GNU General Public License (GPL)  [Reference](https://docs.python.org/3/license.html?utm_source=chatgpt.com) |
| <https://www.postgresql.org> | Database used for backend | Licensed under PostgreSQL License, similar to the BSD or MIT licenses  [Reference](https://www.postgresql.org/about/licence/) |
| <https://ethereum.org> | Public Blockchain (L1, provides decentralized network for L2 Blockchain | Terms of Use provide the terms and conditions  [Reference](https://ethereum.org/en/terms-of-use/) |
| <https://polygon.technology> | Public Blockchain (L2, provides decentralized network for C2) | Terms of Use provide the terms and conditions  [Reference](https://polygon.technology/terms-of-use) |
| <https://arbitrum.io/> | Public Blockchain (L2, provides decentralized network for C2) | Licensed under a Business Source License  [Reference](https://github.com/OffchainLabs/nitro?tab=License-1-ov-file) |
| <https://docs.soliditylang.org> | Smart contracts for managing blockchain transactions through the decentralized blockchain network between all elements of a system of systems   * pragma version 0.8.19 * Forge-std: 1.7.1 * openzeppelin-contracts: 5.0.0 * openzeppelin-contracts-upgradeable: 5.0.0 | Licensed under the GNU General Public License v3.0 (GPL-3.0)  [Reference](https://github.com/ethereum/solidity/blob/develop/LICENSE.txt) |
| <https://web3py.readthedocs.io> | Python software library for interacting with the decentralized network | Licensed under the MIT License  [Reference](https://github.com/ethereum/web3.py/blob/main/LICENSE) |
| <https://deepshield.secublox.com> | Platform service for DeepShield   * platform v0.0.6 * deep-shield-lib version – 0.0.14 |  |

## Platform API

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.

**Step 1**. **Open the Swagger UI**

Navigate to:

|  |
| --- |
| https://deepshield.secublox.com/api/swagger-docs/ |

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2. Authenticate**

* **Click the Authorize button.**

A screenshot of a computer

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

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

**A screenshot of a computer

AI-generated content may be incorrect.**

### Endpoints

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

**Endpoint:**

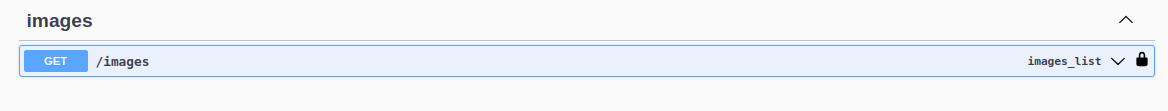
|  |
| --- |
| GET /images |

#### 

#### **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

A screenshot of a computer

AI-generated content may be incorrect.

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

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

A black and white rectangular sign with black text

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

**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:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Examples Response:**A screenshot of a computer program

AI-generated content may be incorrect.

#### **JSON Response:**

|  |
| --- |
| "id": 83, #Unique ID  "name": "ai image", #Image name "creator\_account": {  "username": "iabg\_cc\_app\_1", # app/device username  "id": 32, # app/device id "name": "iabg\_cc\_app\_1", # app/device name "wallet\_url":"https://sepolia.arbiscan.io/address/0x6841E3AEFB55Eb087F923fd2971Bc642b858F79b" # app/device wallet address URL  },  "i\_id": "I0083", # Image ID  "meta\_data\_link": "https://deepshield.mypinata.cloud/ipfs/QmZumGEqkAW8ZN3gcesDk43eyVsBxBtjpgsyaH7yj4xLpY?pinataGatewayToken=cJi7ejhGVyFet2kGKZLUPEWnOnqHe8KHbgN5z8KP3bEoOQEw6nQ-mEsQAELK6AWd", # Image meta data link "signing\_method": "watermark", # This is signing method  "image\_type": "WEBP", # Image extension extract based on the image "created\_at": "2025-02-28", # Image create date "Image": BASE64 data # Image is decrypted into base64 formate "status": { "DeepFake": None, "AI-generated": None }, # This field show image is generated by AI or DeepFake  "watermark": "False", # Check watermark is present in image or not "assign\_account": {  "username": "iabg\_creator\_1", #Creator account username "id": 30, # Creator account id  "name": "iabg\_creator\_1", # Creator account name  "wallet\_url": "https://sepolia.arbiscan.io/address/0xBA4d9B240F88856E70D25AAce 76F99878d1cC947" #Creator account wallet address URL },  "encryption\_type": "with AES 256 key", # Image is encryption by "image\_view": "https://deepshield.mypinata.cloud/ipfs/QmNnbzVypupn87R5sZ1pDzznPtYeCrSzESURzEMZuwzSuF?pinataGatewayToken=cJi7ejhGVyFet2kGKZLUPEWnOnqHe8KHbgN5z8KP3bEoOQEw6nQ-mEsQAELK6AWd" # Image url after encryption |

### 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:**

A screenshot of a computer

AI-generated content may be incorrect.

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

A black screen with green lines

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

click on **Execute** button

A screenshot of a computer

AI-generated content may be incorrect.

#### **JSON Response:**

|  |
| --- |
| {  "status": "success", # Data fetch successfully  "message": "Data fetched", # Message  "data": {  "register": true, #True means image is registered/False means image is not registered  "verify": false,#True means image is verify/False means image is not verify  "register\_hash": "https://sepolia.arbiscan.io/tx/0x3492da84837062f9771d936f06e22a5a3820b6955b630d43a3544d4677eba7f9", # image Register transaction hash  "verify\_hash": "https://sepolia.arbiscan.io/tx/0xNone"# image verify transaction hash is none  }  } |

### 

### 

### 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:**

A screenshot of a computer

AI-generated content may be incorrect.

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

A black screen with green lines

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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

A screenshot of a computer

AI-generated content may be incorrect.

#### **JSON Response:**

|  |
| --- |
| {  "status": "success", # Image register transaction execute successfully  "message": "Registered" # image is registered } |

### 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.python.org [↑](#footnote-ref-2)
2. https://docs.arbitrum.io/build-decentralized-apps/public-chains [↑](#footnote-ref-3)