Visual content verification in blockchain environments

Alexandre C Moreaux, Mihai P Mitrea

IFSA Publishing

Blockchain and Cryptocurrency

Published by IFSA Publishing, S. L., 2023

Visual Content Verification in Blockchain Environments

A. Moreaux and M. Mitrea

SAMOVAR Laboratory, Telecom SudParis – Institut Polytechnique de Paris, Palaiseau, France

E-mail: mihai.mitrea@telecom-sudparis.eu

Received: 29 April 2023 Accepted: 5 June 2023 Published: 11 September 2023

Abstract: We study the use of visual content tracking applications in conjunction with blockchain solutions. The issue consists of multimedia technologies such as fingerprinting requiring the computing power of off-chain machines to operate. We present a workflow using a load-balancing architecture that allows for such operations to be computed off-chain yet benefitting from the level of integrity brought by blockchains. We thus enable the creation of blockchain-backed data storages where multimedia inputs can be identified by their semantic content. We also address the challenge of minimizing computing resources which occurs when on-chain processing is in context, ensuring such an architecture can be deployed and used on state-of-the-art environments. We illustrate our workflow on the Tezos and Ethereum platforms and provide two implementations, relying on an image filter detection use case (featuring the International Standard Content Code), and a museum IPR use case (featuring a robust video fingerprinting method), respectively. We thus demonstrate the mutually beneficial association of on-chain and off-chain applications for multimedia content tracking as well as the blockchain-agnostic nature of the advanced solution.

Keywords: Blockchain, Ethereum, Fingerprint, Interoperability, Load-balancing, Tezos, Verified token, Visual content.

1. Introduction

This paper falls under the scope of the relationship between blockchain and multimedia technologies.

On the one hand, blockchains are peer-to-peer anonymous networks of nodes producing a sequence of cryptographically linked blocks, containing information about the transactions that have occurred in that network [1]. They mainly act as a trusted third party in the exchange of assets and information between untrustworthy actors. Since their inception, blockchains have evolved to support a large area of applicative domains thanks to automated pieces of code called Smart Contracts. Smart Contracts are written in different languages for different blockchains (e.g., Solidity for Ethereum) and run exactly as they are programmed, with no possibility of change or influence from any central authority. Decentralized applications use Smart Contracts as backends serving frontend user interfaces to offer a wide array of services, including but not restricted to, decentralized finance (DeFi) or marketplaces. The digital assets that are meant to be owned and exchanged are referred to as tokens, and can be fungible (interchangeable and splitable, as per legal tender) or non-fungible (representing unique assets and being undividable). Non-Fungible Tokens (NFTs) often serve as the representation of digital art and constitute a 4 billion USD market in 2021 [2]. Yet, the environment is riddled with fraudulent content. The biggest NFT selling platform in the world [3], Opensea, observed that over 80% of the assets being flagged as plagiarized works, fake collections, and spam were created with their simplified “lazy minting” process, accessible to all [4]. This serves as an example to illustrate that NFT abuse is easy, accessible, and rampant [5].

On the other hand, multimedia content represents one of the highly valuable assets on the market today. From video content for cinemas to audio analysis for military applications, nearly every sector benefits from advancements in multimedia content services. Being an asset so valuable, its protection is naturally at very high stakes, be it in academic or industrial settings. Various approaches allow to control the flow of data by hiding it (data encryption), identifying its owners (digital signatures), or tracking the content itself (digital watermarking and fingerprinting). Specifically, near-duplicated content protection (also referred to as visual fingerprinting) is a technology able to identify slightly modified versions of visual content (image or video).

Visual fingerprinting does not feature any intrinsic trust property and blockchain is an appealing solution to this problem. In fact, coupling blockchain to multimedia content presents no conceptual contradictions. Yet, the association between the two is drastically limited by the lack of methodological bridges, as multimedia content processing is a priori prohibitively complex to be executed on-chain.

This paper constitutes an extension of our previous study presented in [6]. It couples the above two notions into an architecture enabling the creation of data storages (e.g., databases) of semantically verified content. The main contribution is a blockchain agnostic, load-balancing workflow that combines the trust and integrity of blockchain to the fingerprinting-based precise content identification to produce verified data that can be stored or minted into tokens. According to this workflow, each time a piece of multimedia content is candidate to be registered into a blockchain-authenticated storage, its semantic content is compared to the previously entries. The authenticated storage can then be updated according to the result of the comparison, and digital assets can be created. Proof-of-concepts of this workflow are provided for the Tezos and Ethereum blockchains.

The present paper is focused on providing the governance mechanism allowing the accommodation of fingerprinting operations in blockchain applicative environments. As such, fingerprinting methods themselves, database exploitation, and security concerns are directly inherited from the state-of-the-art and are out of our research scope.

This paper elaborates on the topic with the following organization. We discuss the current state-of-the-art of blockchain-assisted applications in Section 2 before introducing our methodology in Section 3. In Section 4, we implement the workflow for various blockchain environments and use cases before subsequently analyzing its performance in Section 5. Finally, we conclude and discuss future work in Section 6.

In a nutshell, the main contributions with respect to our previous conference paper [6] are:

- Reconsidering and extending the architectural basis to accommodate the specific needs of different blockchain environments,
- Evaluating the advanced method and its performances in a new blockchain environment, namely Tezos,
- Demonstrating that the concepts brought forward are blockchain agnostic, thus acting at the level of a conceptual blockchain interoperability tool.

2. State-of-the-art

We begin by presenting concise state-of-the-arts of both Smart Contracts (Section 2.1) and visual fingerprinting (Section 2.2) as they are the fundamental notions we shall rely on for the rest of the paper. We then go over their simultaneous uses in current literature (Section 2.3). A lot of the cited studies served as a base for the original source of this extension, [6], as detailed hereafter.

2.1. Smart Contracts

Although theorized in 1994 [7], the concept of Smart Contracts, or software-based automatic contracts, gained popularity with Ethereum [8]. The capacity to enforce agreements between parties without the involvement of a trusted third-party enabled Smart Contracts to gain massive traction in the DeFi and notarization fields, as summarized in [9]. Although legal gray zones and security threats undermined the boom, Smart Contracts quickly spread to other use cases (healthcare, cloud computing, energy, etc.) and the activity of scientific literature in the field suggests that opportunities are still being investigated for various industries [10]. Smart Contracts are also often used as the backend to decentralized applications (dApps e.g., exchanges, marketplaces, etc.) in which case they interact with an off-chain frontend User Interface. More specialized approaches, limited by the computing capabilities of blockchains, tended to use Smart Contracts as complements to legacy applications such as wireless systems [11].

2.2. Visual Content Tracking

While data encryption and identification are out of the scope of this study, we shall focus on content tracking. Specifically, we will need content to be tracked via an easily manipulable invariant digest. Although cryptographic hashing accomplishes the above, it can only be used to check for strict differences and not to express said differences. Indeed, an image with a single pixel modified will have a completely different cryptographic hash than its original.

This is important because multimedia content changes during its lifecycle. All common operations (recording, uploading, broadcasting, sharing, downloading, etc.) modify multimedia content in often

imperceptible ways (luminosity, format, compression, etc.). This ties to the concept of near-duplicated content, that covers content that is naturally of maliciously modified. Specifically, [12] defines near-duplicated content (videos in their context) as "Identical or approximately identical videos close to the exact duplicate of each other, but different in file formats, encoding parameters, photometric variations, editing operations, different lengths, and certain modifications.".

We can overcome this issue using fingerprint techniques, which contrarily to cryptographic hashing utilize the semantic content of their input to generate digests that are invariant to near-duplicated modifications. Consequently, inputs that are very similar will generate fingerprints that are close (in terms of a given similarity measure, like normalized correlation or Hamming distance, for instance). This is illustrated in Fig. 1.

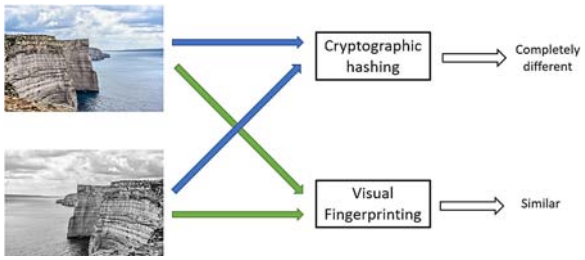

Fig. 1. Invariant digest comparisons in cryptographic hashing vs. similarity preserving visual fingerprinting.

This feature allows fingerprints to be used to find semantic nearest neighbors of an input amongst a given dataset. Their compromise with regards to cryptographic hashing consists in their security features and size. Fingerprinting and cryptographic hashing are tools that simply do not solve the same problem. In this paper, we will use fingerprinting to identify near-copies of visual content and cryptographic functions to shorten invariant digests for easy storage.

2.3. Joint Usage of Multimedia Content Tracking and Visual Fingerprints

When it comes to multimedia content being used in blockchain contexts, Non-Fungible Tokens (NFTs) are a re-occurring concept. NFTs are unique blockchain digital assets that represent data and are created, stored, and exchanged by users and Smart Contracts. As such, digital art and other multimedia asset representations are securely distributed in a massive market projected to reach 200 billion USD by 2030 [2], featuring large and trusted actors [3]. Yet, NFTs are vastly misunderstood in their capacity to represent assets. They are also notoriously lenient with Intellectual Property Right (IPR) ideas, the NFT copy and stolen content market fueling debate and tainting the reputation of the space [13].

Outside of NFTs, multimedia processing itself can be enhanced via blockchain applicative technologies as part of the process [14] or hand in hand with off-chain technology [15]. The joint uses of content protection techniques and blockchains are summarized in [16]. This holistic survey cites encryption, watermarking, and transaction tracking fingerprinting, indicating that near copy detection using visual fingerprinting techniques had not yet been associated with blockchain before [6].

The notion of databases, or more generally data storages, being used alongside blockchains is not novel. For instance, databases and blockchains were used in the IoT use case analyzed in [17] and the cloud computing study in [18]. To the best of our knowledge, the replicated hashed "shadow" on-chain database as an integrity verifier brought forward in [6] was novel. The idea of a load-balancing architecture for blockchain-enhanced applications we used in this paper was brought forth in [19].

3. Methodology

In this Section, we detail the processing workflow we designed for serving the needs of coupling blockchain to visual fingerprinting. We will start by explaining our method in a general sense, before detailing each of the methodological blocs constituting the architecture.

3.1. General Architecture

The processing workflow we advance is supported by a generic architecture illustrated in Fig. 1. It is designed as to ensure the processing and exchange of data amongst four logical entities: an off-chain Database, an off-chain App, a Smart Contract, and a Token Contract. The first three entities represent the pillars of the solution while the Token Contract is called upon the successful processing of an input and is not involved in the inner workings of the solution.

The initial setup of the database and deployment of the Smart Contract is done by a qualified blockchain expert. Once setup, no more blockchain expertise is required and an App operator can use the architecture.

The process starts with visual content being fingerprinted, and these fingerprints being stored on a database. They are then initialized on the blockchain via the Smart Contract, which serves as a pseudo database. This on-chain tamperproof, redundant database allows the Smart Contract to serve as an arbiter ensuring the database has not been tampered with. It intervenes before the App compares an input (be it a new piece of content or a suspected copy) to each of the off-chain database entries. Three results are possible, and are illustrated in the implementation (Section 4, Fig. 12):

- The input is detected as a copy of existing content (i.e., the fingerprint is identical to an entry of the database) and the operator is informed as such and the process stops.
- The input is detected as near-duplicated content, or a *near copy* of one of the entries (cf. Section 4) and the operator may decide to consider the input as original or a copy.
- The content is not detected as the copy of existing entry. The operator can add it to the database by answering a prompt.

Upon its arrival into the on-chain database, the entry is minted as a Non-Fungible Token and sent to the wallet of the initiator of the transaction. This process is illustrated in Figs. 2 and 3.

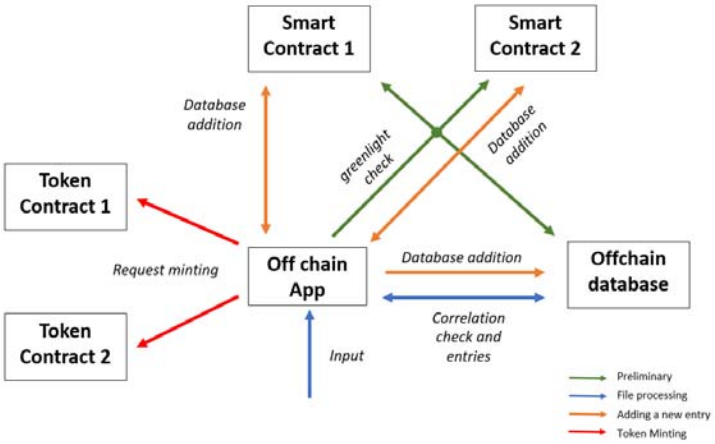

Fig. 2. Advanced architecture, bearing on-chain Smart and Token Contracts and an off-chain App and database.

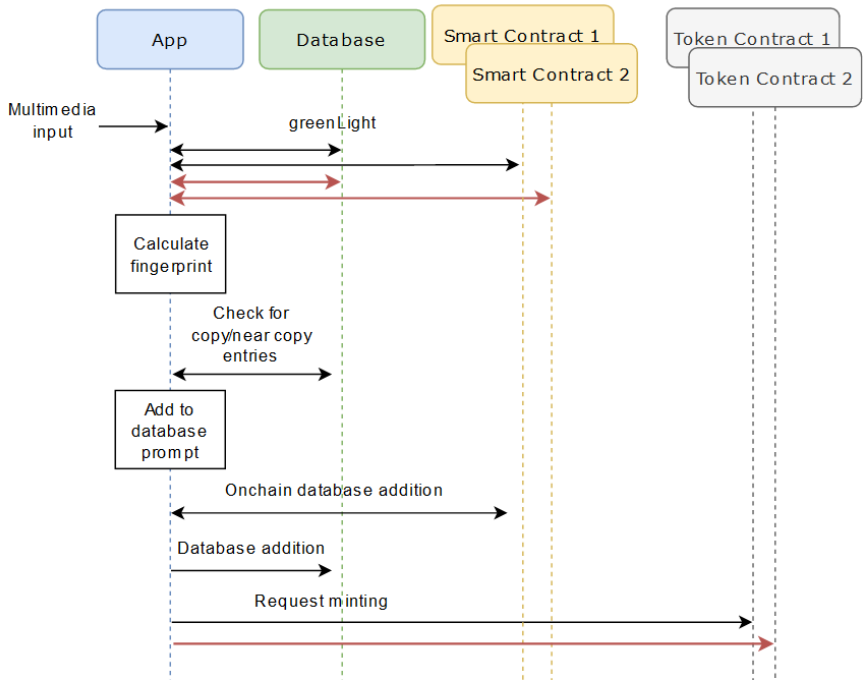

Fig. 3. Step-by-step addition of a new entry in the database, operations related to the second blockchain architecture in red.

3.2. Off-chain Entities

This Section details the two off-chain entities shown in Fig. 2, namely the Database and App.

The advanced architecture does not worry itself with the exact technology managing the database. In fact, it only has light lifting to do, as it only needs to hold the fingerprints of the multimedia content and to pass that information to the App when requested. Although it would be possible to hold the content itself in the database and to fingerprint it upon retrieval, a lighter and more private database allows for faster processing and less potential privacy concerns.

The App has a central role in the process. Not only does it interact with both databases, but it also acts as the only point of contact for the operator. As such, the visual interface can be designed to make the process intuitive and easy to operate. In the context of a proof-of-concept, we did not develop any graphical interface and we interacted with the App by using a command prompt.

The App is given a visual content file (.jpg, .mp4, .pdf, etc. depending on the use case) and an optional threshold (that defaults to a recorded value) as input parameters and begins by establishing a connection with the Smart Contract, invoking a *greenLight* function. This function returns *True*, allowing the process to continue, if and only if the off-chain and on-chain databases match. It does so by requesting the size of the map of hashes from the Smart Contract and using the compare function of the latter (Subsection 3.3) for each one of the entries of the off-chain database. This process is expedited by the fact that the database contains fingerprints that need not be systematically reprocessed. The *greenLight* function returning *False* will interrupt the process and inform the operator that the database has been tampered with.

Once this important control passed, the App computes the input file's fingerprint and compares it to all the entries in the off-chain database. As explained in Subsection 3.1, three possible results are presented to the operator: copy, near copy, or no copy. In the latter two cases, the operator may prompt the App to add the input to the database. The App then transactions the Smart Contract via the deployer wallet to add the hash of the new fingerprint to the on-chain database, before adding the fingerprint (identified by its hash) to the off-chain database. Note that the fingerprint is hashed before being stored in the Smart Contract because of size and format concerns (i.e., they cannot be defined in blockchain development). If the fingerprint in context happens to output short identifiers, the hashing step may be skipped as it is not essential to the proper functioning of the code, although it could still be used to add a layer of privacy to the information.

Once the transaction that added the new entry to the Smart Contract is validated, the App queries the minting of a unique NFT (cf. Subsection 3.3). The entire workflow is shown in Fig. 3.

Although the method used to identify content (i.e., the fingerprinting method) is the core of the application, the general architecture is independent of its specificities. The role of the fingerprinting method is twofold. First, being the initial step of the process, it defines the input format. Indeed, near copy detection has use cases using a variety of data formats (images, video, text, etc.) some of which might focus on semantic content whilst others could include metadata or instance data. Second, the detection can only be as precise as the specific fingerprinting method permits. As opposed to having a universal solution, appropriately selecting a fingerprinting method on a case-by-case basis will yield the best results. For illustration purposes, we elected two complementary methods to illustrate this point [22, 23].

The thresholds used to detect near copies also depend on the use case. If the objective is only to detect very close copies of the content in the database, we would set our normalized correlation threshold close to 1, or our maximum Hamming distance very small (in the range of 0 – 3 bits for a 72-bit identifier). If we are more generally looking to detect the same semantic content after alteration, we would set our normalized correlation threshold between .6 and .8, or our maximum Hamming distance between 8 and 12 (for binary fingerprints of size for a 72-bit identifier). For our Section 4 implementations, with a goal of general detection in mind, we used a threshold of 0.7 for the normalized correlation and a maximum Hamming distance of 10. Of course, these limits can and should be tailored to a given set of circumstances and objectives.

3.3. On-chain Entities

This section details the two on-chain entities shown in Fig. 2, namely the Smart Contract and Token Contract.

As explained in Subsection 3.2 and Fig. 2, the Smart Contract is used on two occasions: to provide information to the *greenLight* function and to process a new entry. The former does not require input data whilst the latter requires a hash and an optional string of general information concerning the entry. It maps these two entities into a structure containing a Boolean to indicate the existence of the hash and an optional string containing general information. In addition, it implements five functions.

Three of these functions are of *get* type and allow to communicate information about the on-chain Database to the App. They return the size of the map, the information associated with a hash, and the Boolean associated with a hash, respectively. The latter serves as the comparison function that is called by the App during the *greenLight* function. The other two functions manage database entries, respectively providing the addition and deletion of entries. The addition function verifies prior inexistence of the entry in the database, indexes relevant information (if present in the parameters), adjusts the size of the map and returns a Boolean to indicate successful processing. The deletion function checks for the existence of the entry and adjusts the size of the map if needed before returning a Boolean. The addition and deletion functions can only be called by the address that deployed the Smart Contract. If a use case requires multiple addresses to call the Smart Contract, a whitelist can replace the "only deployer" approach.

The Token Contract is called upon after the successful addition of a new entry in both databases. It generates tokens that contain a trace of this workflow in their metadata. In our example, the token contains the hash of the original fingerprint and is sent to the deployer wallet as it is the central entity of the use

case. These tokens can then stay in this wallet or be sent manually or automatically to addresses belonging to the Intellectual Property Rights (IPR) holders, for example. It would be simple enough for the operator to indicate the address of the content provider for the token to be distributed directly upon validation. This NFT could be more complex, but its format largely depends on the use case. If this architecture were used to certify content before it is sold as original, one could imagine additional information being present in the token to ensure the good standing of the content the token represents. Such additional information may relate to the transaction number of the initial admission of the entry in the database or the electronic signature of the operator.

Our initial work being on Ethereum, we selected the popular ERC721 standard [20] in which we put the hashed fingerprint of the file. If dealing with Tezos, the FA2 [21] standard can be used instead. Please note that within this proof-of-concept, the burning (or deletion) of the token that was created alongside the inclusion of the entry in the database does not occur.

4. Experimental Illustration

In this Section, we walkthrough and discuss illustrative implementations of the workflow and architecture brought forth in Section 3. Access control not being a central feature of this paper, we set out two parties (1) `creator`, which initializes the Smart Contract and has all access rights; and (2) `otherUser`, which only has the basic view right default to all blockchain users and cannot modify the Smart Contract.

We start by going through a basic implementation of the Smart Contract and its use through a scenario on the Tezos infrastructure (in Section 4.1) before moving on to two complete implementations of the entire workflow using the Ethereum framework (in Sections 4.2, 4.3 and 4.4). Of course, the Ethereum Smart Contract will accomplish the same things as the one detailed in Subsection 4.1., and the infrastructure (presented in Sections 4.2, 4.3 and 4.4) logically functions just as well with a Tezos Smart Contract.

4.1. Smart Contract Implementation

The Smart Contract implementation is explained and illustrated for the Tezos platform using SmartPy [24], an online IDE for Tezos Smart Contracts available through a Python library. SmartPy provides us with clear test scenario capabilities, allowing us to illustrate the use of the Smart Contract to readers unfamiliar with Smart Contract development. Tezos development is based on meta-programming: as such, the code we write is not directly run but serves to construct the actual Smart Contract that will run on the blockchain [25].

This Subsection will focus solely on the Smart Contract, using a simple, nondescript, illustrative scenario. Figs. 4 and 5 show the SmartPy functions, which are lightweight and intuitive, as required by blockchain development.

```
def __init__(self, owner):
    self.init(
        fingerprintHashes = sp.map(tkey = sp.TString), owner=owner
    )

@sp.entry_point
def addToDB(self, info, hash):
    sp.verify(sp.sender==self.data.owner, 'Only owner.')
    sp.verify(self.data.fingerprintHashes.contains(hash)==False)
    self.data.fingerprintHashes[hash] = sp.record(info=info)


@sp.entry_point()
def deleteEntry(self, hash):
    sp.verify(sp.sender==self.data.owner, 'Only owner.')
    del self.data.fingerprintHashes[hash]
```

Fig. 4. The SmartPy Smart Contract entry point functions which manage the on-chain database.

```
@sp.onchain_view()
def compare(self, hash):
    sp.result(self.data.fingerprintHashes.contains(hash))

@sp.onchain_view()
def getSize(self):
    sp.result(sp.len(self.data.fingerprintHashes))

@sp.onchain_view()
def getInfo(self,hash):
    sp.result(self.data.fingerprintHashes[hash].info)
```

Fig. 5. The SmartPy Smart Contract view functions which pass on information the Smarty App.

The two functions shown in Fig. 4 (excluding the constructor __init__) will manipulate the map of hashes sent by the app in a complete use case. The first adds its hash parameter alongside some information after verifying the request is sent by the creator and that the hash is not recorded previously. This last check can be removed if overriding were allowed. The second function simply deletes an entry if requested by the owner. These are the only two functions that write information on the blockchain (hence the *@entry_point*).

The three functions shown in Fig. 5 are the *get* functions which pass on information to the App. They only read data and are hence preceded by *@onchain_view*. They send back the existence of a given hash in the map, the size of the map, and the info associated with a given hash, respectively. Now, we build a test scenario that will use this Smart Contract. Its initialization is shown in Fig. 6.

```
@sp.add_test(name = "FingerprintStorage test")
def test():
    scenario = sp.test_scenario()
    creator = sp.test_account("creator").address
    otherUser = sp.test_account("otherUser").address
    c1 = FingerprintStorage(owner=creator)
    scenario += c1
```

Fig. 6. The SmartPy test scenario initialization with two users, having full rights and no rights, respectively.

We begin by setting up the scenario and the accounts we will use (`creator` and `otherUser`) so we can reference their addresses. We then attempt to initialize the on-chain database as the App would do, which is illustrated in Fig. 7.

```
c1.addToDB(hash="0x0001", info="first input").run(sender=creator)
c1.addToDB(hash="0x0001", info="same input").run(sender=creator, valid=False)
c1.addToDB(hash="0x0002", info="unauthorized user").run(sender=otherUser, valid=False)
c1.addToDB(hash="0x0002", info="second input").run(sender=creator)
c1.addToDB(hash="0x0003", info="third input").run(sender=creator)
scenario.show(c1.getInfo("0x0003"))
```

Fig. 7. The SmartPy test scenario on-chain database initialization, seen from the Smart Contract.

When `creator` requests the addition of a new, properly formatted input the transactions are executed without issues, as done in the first line of Fig. 7. On the contrary, inputting a previously recorded entry (line 2, Fig. 7) and `otherUser`'s attempts (line 3, Fig. 7) are rightfully unallowed operations, and hence reverted (denied by the blockchain). At the term of these operations, the Smart Contract's storage matches what is shown in Fig. 8.

FingerprintHashes		Owner
Key	Info	tz1aRxdK2bTSgp…
'0x0001'	'first input'	
'0x0002'	'second input'	
'0x0003'	'third input'	

Fig. 8. SmartPy Smart Contract storage after the addition of the three inputs shown in Fig. 7.

The addition of new inputs can be prone to errors, which we illustrate in Fig. 9, where an unwanted entry is added then deleted by `creator`, whilst `otherUser` is shown to be unable to affect said entry.

```
c1.addToDB(hash="0x9999", info="input including a mistake").run(sender=creator)
c1.deleteEntry("0x9999").run(sender=otherUser, valid=False)
c1.deleteEntry("0x9999").run(sender=creator)
```

Fig. 9. The SmartPy test scenario showing an unwanted entry being added and subsequently removed.

At the term of these transactions, the storage is back to the state shown in Fig. 8. We consider this storage to correspond to the initial storage of a given use case. When the App executes the *greenLight* function, the operations shown in Fig. 10 would be requested.

The App would start by checking the number of recorded hashes is indeed equal to the number of entries it has in its local storage (3 in this case) and would then use the *compare* function to check the hashes of our recorded fingerprints appear on the blockchain. If this test passes, the *greenLight* function returns *True* thus ensuring that the off-chain and on-chain databases match.

```
scenario.verify(c1.getSize() == 3)
scenario.show(c1.compare("0x0001"))
scenario.show(c1.compare("0x0002"))
scenario.show(c1.compare("0x0003"))
```

Fig. 10. The SmartPy test scenario *greenLight* execution, seen from the Smart Contract side.

The Smart Contract used in the Subsection is available on SmartPy via: smartpy.io/ide?cid=QmZa8H7PPHDXtmEk5umZqFsSS5R4wBmLcTr7WxDSvNqHwk&k=24deab4dc14ccfda82ca. The test scenario can be ran using the "Run Code" button at the top left of the screen.

The Smart Contract was also deployed to the Ghostnet testnet at the following address: KT1MSVoHdoYQpWPBXw4QbfvjSU1abizJbzM2. Its functions, storage, deployment figures, etc. can be searched for using a Tezos explorer, such as TzKT [26].

4.2. Introduction to the use Cases

This paper studies two use cases relating to ensuring IPR for a museum's virtual visit (cf. Subsection 4.3), and a filter based near-copy detection of mirflickr25k [27] images (cf. Subsection 4.4), respectively. These use cases follow the architecture put forth in Fig. 2 and their implementations are supported by the Ethereum framework.

We consider a 3-node, Hyperledger Besu EEA (Enterprise Ethereum Alliance [28])-compliant Proof-of-Authority private blockchain deployed on an Amazon Web Services sever, as well as on the now deprecated Rinkeby Ethereum testnet accessed through the Infura node cluster. We interacted with the Smart Contracts using the web3py library [29].

To test these implementations, we will compare the entries in the database to artificially created (more or less) near copies. We created modified versions of the original inputs by subjecting them to standard image processing attacks, namely: conversion to black and white, brightness increases, cropping (50%), JPEG compression at a quality factor of Q = 90, resizing to 600x400, and combinations of these alterations. Fig. 11 shows images appearing both use cases before and after such modifications. In this example, they were subjected to cropping and brightness increase, and to resizing, respectively.

These modified versions are given to the architecture as inputs. Naturally, some of them will be semantically so close to one of the entries that it will be categorically refused by the App, whilst others bear so little resemblance to originals that they will be considered as original inputs. The border between

these answers of course lies in the selection of the threshold decided on for each situation.

Given that the detection performance solely relies on the specific fingerprinting method in use, and that the architecture put forward in this paper has no effect on the performances of said method, we will not dwell on them here. Extensive performance analysis for these respective methods is available in [22] and [23] and were corroborated by our tests.

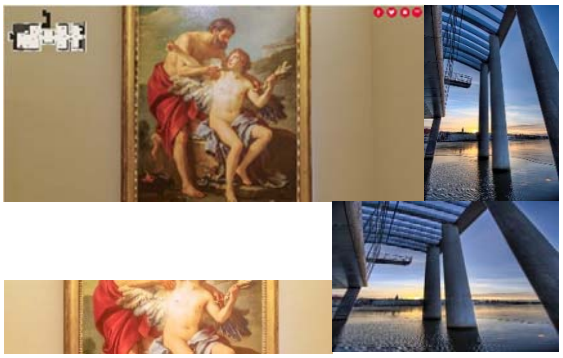

Fig. 11. The before (top) and after (bottom) of images from the use cases (museum IPR on the left, mirflickr25k on the right) having been subjected to image processing attacks.

4.3. Museum Virtual Visit use Case

The first use case we implemented is set up to simulate a museum wary of multimedia content posted online being copied. We used a database comprised of sequences extracted from the virtual visit of six rooms offered by the Louvre Museum in Paris during the COVID-19 pandemic [30]. We used these images for strictly academic and non-commercial purposes and do not intend any infringement of the Louvre's IPR. Test videos were sampled to 1 frame per second, the fingerprints were computed according to [22] and were compared by using normalized correlation.

We find ourselves in the first scenario where six image sequences are fingerprinted in the database and the Smart Contract has previously been deployed on the Rinkeby testnet alongside the Token Contract. Fig. 12 shows the results of us giving one of the original videos as an input, whose fingerprint appears as is in the database, as well as a near-copy case. We altered the sequence of another original video by cropping the top and bottom 25% of each image and increasing their luminosity (as illustrated in Fig. 11) before feeding it to the App as a new input. Both results can be seen in Fig. 12 and show appropriate behavior, indicating the copy and detecting the near copy, respectively.

We then compile a random modified sequence of images from the different inputs to create a sequence that has no significance to the original database. If we run the App using this new sequence as an input, we get a prompt, illustrated in Fig. 13, which indicates the sequence is considered semantically new with respect to the database. If the operator wishes to add this input to the database, they may accept this prompt which transactions the Smart Contract, and in turn the Token Contract. The ensuing transactions hashes are shown in Fig. 13 and may be cross checked using a Rinkeby explorer such as [31]. The ERC721 token minted for the occasion is sent to the wallet that deployed the Smart Contracts.

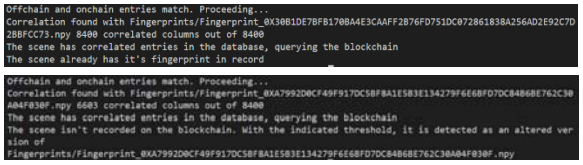

Fig. 12. The App's responses to being fed a copy of one of the database entries (top) and a near-copy of one the database entries (bottom).

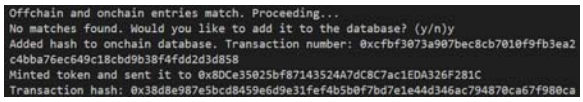

Fig. 13. The successful addition of a new entry (detected as original) in the database and its subsequent tokenization.

If a malicious user were to gain access to the database and delete an entry from the records for their own entry to be perceived as semantically original, the *greenLight* function would not permit the App to function. We acted as such a user and the result is shown in Fig. 14. The same modification cannot happen with the verification database, as is appears on-chain and is subsequently unalterable.

Fig. 14. The answer to the App being run after the database has been tampered with i.e., a *greenLight* function failure.

The Smart Contract and Token Contract we used for these tests can respectively be found at *0xA75f207314C85F4891657a2D4f73b19b88b21dc9* and *0xAAFFfFF06a971b57ca87953010135d771B91f965* (their information is available and browsable through a Rinkeby explorer).

4.4. mirflickr25k use Case

This use case serves to illustrate the variety of possible inputs and identification means compatible with this workflow. It features a more generic database identified using International Standard Content Codes (ISCC) [23]. The ISCC is a full ISO/AWI 24138 standardization work item whose goal is to provide an

open-source, cross-sector, universal identifier of different kinds of content. For our purposes, it is also a lightweight and similarity preserving fingerprinting method. ISCC codes are composed of 4 parts: their Metadata code, Content code, Data code, and Instance code. For this proof-of-concept implementation, we focused on the Content code portion of the fingerprint, although other use cases could very well take full advantages of the different facets it offers. In fact, our architecture can not only compare full ISCC codes, but make decisions based on separate processing. For instance, we could require a strict threshold of differences between Content codes while enabling a looser check for Metadata and Instance codes to differentiate original content brought forth by other users or with different encodings. The partial content flagging feature of the ISCC could also be put forth to identify copied content being used within other content. This feature is illustrated in an infographic available on the ISCC's website [23] and shown Fig. 15. Using this feature, near copy detection becomes even more important as content inserted in other content is naturally modified.

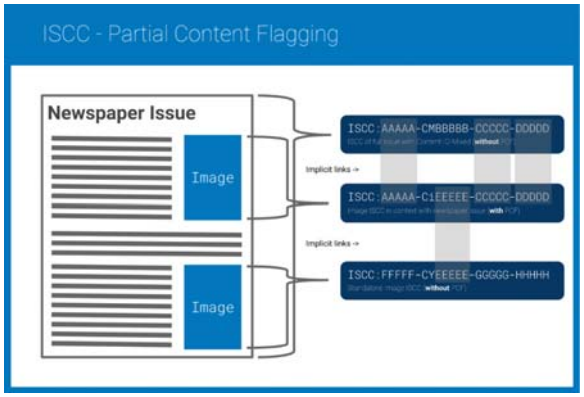

Fig. 15. Illustration of the content flagging feature of the International Standard Content Code [23].

The inputs we used are represented by a collection of JPG images of various sizes taken from the mirflickr25k set [27].

An interesting feature of the ISCC is its capacity to generate similarly formatted outputs from completely different input formats. The media identifier is somewhat universal, and could enable databases of images, text, video, audio, etc. to be treated uniformly. In fact, the ISCC was designed for blockchain-based registration, it is as such short (between 13 and 55 characters), so we forego the hashing of the code and store the code directly in the Smart Contract.

Just as in the first use case, we submitted three types of content to the architecture: exact copies of content it was already holding, modified versions of said entries using the multimedia attacks presented in Section 4.3, and completely unrelated semantic content (in this example other mirflickr25k images).

When being run, the App acts the same as with the first use case, and the responses shown in Figs. 12, 13, and 14 can be observed. Only the inputs and detection method are modified.

5. Discussions on Workflow Performances

In this Section, we account for the workflow presented in Section 3 and the implementations provided in Section 4 to discuss the specificities of the workflow. We start by noting the genericity of the architecture (Section 5.1) before taking a closer look at resource consumption (Section 5.2) and interoperability (Section 5.3).

5.1. Workflow Genericity

The workflow advanced in the paper provides a robust structure catering to the needs of verifiable data integrity. It does so by making the best of the mutually beneficial association of on-chain and off-chain technologies. It also enables the variable processing of multiple forms of multimedia content. As demonstrated in Section 4, the fingerprinting method used in the workflow can easily be modified without affected the other parts of the algorithm. As such, one could use this methodology to track photography being copied with filter modifications just as well as tracking changes of metadata in audio files.

As stated in the introduction, this paper analyses the governance mechanism of the proposed architecture. As such, we will not provide detailed reports on near-copy detection performance (as it is purely determined by the fingerprinting methods themselves) nor on security mechanisms. Briefly:

- The architecture does not impose supplementary constraints to multimedia processing and as such the analysis provided by the developers of the specific methods stays relevant.
- We take advantage of the native security features brought by the blockchain environment. As such, the advanced workflow can only be as secure as the link between app and blockchain.

5.2. Resources Usage

This section investigates the overall computational load required by our workflow.

Although the App itself requires computation, it is naturally substantially inferior to the computation of the Smart Contract. The former is also largely dependent on the fingerprinting method in question. Hence, we focus on the blockchain computational load.

Computation on the blockchain can be measured with gas. The gas fee is defined by the cost (in local cryptocurrency) users pay validators of the blockchain. Each transaction costs a certain amount of gas, and a given block has gas limits that restrict what can be executed within a block. We also must keep in

mind that execution times and gas costs will vary significantly depending on the blockchain in context as well as the network traffic at the time of execution, as these costs are dynamic. As such, the results we provide here are linked to their context and execution conditions.

First, the Smart Contracts are simple and hence lightweight. The total storage of the Smart Contract is decided upon deployment and can be estimate via diverse tools such as Tezos' client-server protocol (RPC). This helps generate values for fees, gas limits, and storage values. These automatically generated amounts, as well as the deployment figures that can be found on an explorer are shown in Fig. 16. They confirm the design philosophy of the workflow is respected by remaining small. Of course, practical applications need to account for data they will store as hashes in the Smart Contract and add that value to the storage limit of their code. At the time of writing, *tez* (the native Tezos token) is worth just around one USD, making this deployment cost less than 22 cents. Implementations accounting for storages would indeed cost more, but stay very affordable, especially on blockchains with low gas costs such as Tezos.

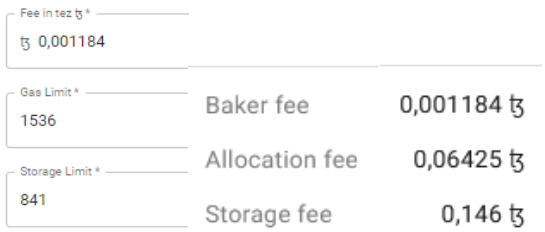

Fig. 16. Tezos Smart Contract RPC estimates for deployment (left) and deployment fees (tight).

When it comes to our Ethereum implementations, the detail of the initial deployment of the on-chain programs can be found in Fig. 17 (the deployment of the Smart Contract is the same for both use cases). It shows single block deployments of the Smart and Token Contract, respectively using 15.24% and 61.45% of gas limits (set by default at 4.5 million), for a total of 0.03451321ETH (for a gas price of 10 Gwei, or 10^{-8}ETH). Use cases not needing the tokenization of their assets can eliminate the latter an only use a single lightweight Smart Contract.

For the museum IPR use case, populating the Smart Contract with 6 entries cost us 0.000114ETH per entry, whilst the tokenization cost 0.000226 ETH per entry (for a gas price of 1.5 Gwei). Although this step is the biggest resource sink in the entire process, it stays in the scope of a blockchain application. The gas and time spent scales linearly with the number of entries, so even databases of a few hundred to a few thousand entries could comfortably be processed in the span of a couple of hours. This of course depends on the block rate of the blockchain in context.

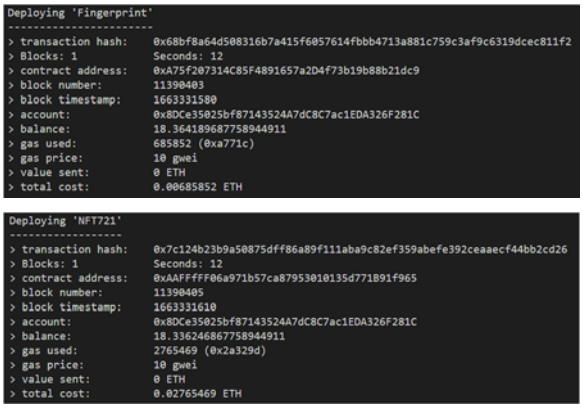

Fig. 17. Deployment figures of the Ethereum Smart Contract (top) and Token Contract (bottom).

After the setup and for general use, the Smart Contract is only invoked at two specific moments. This leaves most of the processing up to the faster and more efficient app. The first use is the *greenLight* function. This instance does not constitute a transaction as it does not write any information on the blockchain. This call does not cost gas and is not limited by slow block rates. In our experience and with our testing setup, this step never added more than 2 seconds of execution to the processing of an input. The second use is in case a new entry is to be added to the database. This step is essentially the initial setup brought to the scale of a single entry. In fact, the transaction we executed to illustrate Subsection 4 cost the same amount of 0.000114ETH. As was our aim, this localized and minimal use of blockchain enables us to avoid long processing times and excessive gas fees.

5.3. Interoperability

The notion of interoperability appears in diverse aspects in this workflow. Not only is the infrastructure interchangeable, but the applicative components are too. This enables the solution to be used together with state-of-the-art components from different sectors.

In terms of hardware, the off-chain database can be a simple one as a cloud computing data storage, implementing no features or modern customization, access management, and data integrity. The same applies to the blockchain. Only minimal tools are used, and as such these Smart Contracts can be transposed to any modern application-oriented blockchain. Although the language and specific performances in gas and time will differ, they stay relatively uniform. In fact, we notice that the code is formatted very similarly in Solidity (Ethereum) and SmartPy (Tezos), that processing times are in the order of seconds for both (mostly affected by blockrates), and that gas costs remain low given the simplicity of operations required.

When it comes to the software, the only non-trivial requirement for the programming language is the

existence of a library connecting to the desired blockchain. For instance, web3py [29] allows one to connect to the Ethereum blockchain via Python, web3.js [32] is its JavaScript counterpart, and Taquito [33] is a TypeScript Library that enables Tezos interactions. The requirements on the fingerprinting technology are even more lenient, as its input can be formatted to the desired length via a cryptographic hashing function. The output information can also be formatted to suit any token standard, making the Token Contract a flexible methodological brick as well.

6. Conclusions and Future Work

In this paper, we detail a framework and workflow enabling multimedia content tracking to be backboned by blockchains. We not only use a load-balancing architecture to enable the complex computation to even be possible in such environments but provide a mutually beneficial relationship between the applicative bricks available on-chain and off-chain. This method also makes the most of the flexibility of its components to host state-of-the-art fingerprinting techniques, never restricting their features and performances. These advantages lead this architecture to enable robust semantic data verifiability. As such, this verified data can further be used in a large array of contexts, including ones that require strong Intellectual Property features. Additionally, the methodology can seamlessly be exported to various blockchain environments and use various database technologies.

Further work could investigate and develop compatible Smart Contracts in new environments, examine and reinforce the innate security of the system, or extend the Token Contract's functionalities to make the best out of the verification process brought by the workflow.

Acknowledgement

We acknowledge Titusz Pan and Sebastian Posth from the ISCC foundation for out fruitful exchange leading to the integration of ISCC into this methodology. We acknowledge Najah Naffah for his insights in applicative blockchain environments.

References

[1]. S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[2]. Grand View Research portal (https://www.grandviewresearch.com/industry-analysis/non-fungible-token-market-report)

[3]. DappRadar Web portal (https://dappradar.com/nft/marketplaces)

[4]. Opensea Twitter post (https://twitter.com/opensea/status/1486843204062236676)

[5]. D. Das, P. Bose, N. Ruaro, C. Kruegel, G. Vigna, Understanding Security Issues in the NFT Ecosystem, in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, November: 2022, pp. 667–681.

[6]. A. Moreaux, M. Mitrea, Blockchain Assisted Near-duplicated Content Detection, in *Proceedings of the 1st Blockchain and Cryptocurrency Conference (B2C' 2022)*, Barcelona, Spain, Nov 2022, pp. 98-103.

[7]. N. Szabo, Formalizing and Securing Relationships on Public Networks, *First Monday*, Volume 2, Issue 9, 1997.

[8]. V. Buterin, Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform, *White Paper*, 2014. https://ethereum.org/

[9]. V. Dhillon, D. Metcalf, M. Hooper, Blockchain Enabled Applications, *Apress*, 2017.

[10]. T. Hewa, Y. Hu, M. Liyanage, S. Kanhare, M. Ylianttila, Survey on Blockchain-Based Smart Contracts: Technical Aspects and Future Research, *IEEE Access*, 2021.

[11]. X. Li, P. Russell, C. Mladin, C. Wang, Blockchain-Enabled Applications in Next-Generation Wireless Systems: Challenges and Opportunities, *IEEE Wireless Communications*, Vol. 28, No. 2, April 2021, pp. 86-95.

[12]. Liu, J., Huang, Z., Cai, H., Shen, H. T., Ngo, C. W., and Wang, W., Nearduplicate video retrieval: Current research and future trends, *ACM Comput. Surv*., 45, 4, 2013, Art. No. 44.

[13]. Last Week Tonight, Cryptocurrencies II. (https://www.youtube.com/watch?v=o7zazuy_UfI)

[14]. R. Li, Fingerprint-related chaotic image encryption scheme based on blockchain framework, *Multimedia Tools and Applications*, Vol. 80, Issue 20, 2021, pp. 30583–30603.

[15]. F. Frattolillo, A Watermarking Protocol Based on Blockchain, *Applied Sciences*, Vol. 10, Issue 21, 2021, 7746.

[16]. A. Qureshi, D. Megías Jiménez, Blockchain-Based Multimedia Content Protection: Review and Open Challenges, *Applied Sciences*, Vol. 11, Issue 1, 2021.

[17]. L. Tseng, X. Yao, S. Otoum et al., Blockchain-based database in an IoT environment: challenges, opportunities, and analysis, *Cluster Computing*, Vol. 25, 2020, pp. 2203-2221.

[18]. X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, L. Njilla, ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability, in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 468-477.

[19]. M. Allouche, M. Ljubojevic, M. Mitrea, Visual document tracking and blockchain technologies in mobile world, in *Proceedings of the Electronic Imaging, International Conference on Imaging and Multimedia Analytics in a Web and Mobile World (EI 2021)*, January 2021, Online, France, pp.279:1-279:7.

[20]. W. Entriken, D. Shirley, J. Evans, N. Sachs, EIP-721: Non-Fungible Token Standard, *Ethereum Improvement Proposals*, No. 721, January 2018.

[21]. FA2 A unified token contract interface, *Tezos*, https://tezos.b9lab.com/fa2

[22]. A. Garboan, M. Mitrea, Live camera recording robust video fingerprinting, *Multimedia Systems*, 22, 2016, pp. 229–243.

[23]. International Standard Content Code Foundation portal, https://iscc.foundation/iscc/

[24]. SmartPy Tezos IDE. (https://smartpy.io/)
[25]. SmartPy, 'Meta-Programming'. https://smartpy.io/releases/20220607-708da61f52c9d66c88f593ffc2915c52545d6090/docs/introduction/meta_programming/
[26]. TzKT: Tezos blockchain explorer, https://tzkt.io/
[27]. Mirflickr25k dataset, https://www.kaggle.com/datasets/paulrohan2020/mirflickr25k
[28]. Ethereum Entrprise Alliance Specification portal, https://entethalliance.org/technical-specifications/
[29]. Web3py documentation, *Web3Py*, https://web3py.readthedocs.io/en/v5/
[30]. Le Louvre online tour portal. https://www.louvre.fr/en/online-tours
[31]. Rinkeby Testnet- Etherscan. https://www.rinkeby.etherscan.io
[32]. Web3.js – Ethereum JavaScript API, https://web3js.readthedocs.io/en/v1.8.2/
[33]. Taquito TypeScript library, https://tezostaquito.io/

Published by IFSA Publishing, S. L., 2023