A Blockchain-Based Framework for Secure Digital Asset Management.

Vu Tuan Truong and Long Bao Le

***Abstract*—In the current age of digital world with the emergence of metaverse, digital assets are increasingly recognized and become more and more valuable. Unlike real-world assets, managing digital contents is more challenging since their associated information might be leaked widely on the Internet, making them worthless. Traditional digital asset management (DAM) systems based on third-party authorities and centralized databases have various weaknesses, threatening the benefits of stakeholders. In this paper, we propose a blockchain-based DAM framework utilizing smart contract, InterPlanetary File System (IPFS), and multi-layer encryption mechanisms for access control of digital assets in the metaverse. Our proposed design eliminates the intervention of intermediaries and offers a wide range of advanced security features such as resistance against data leakage and data alteration without trust assumptions among participants. Besides, key features of blockchain are leveraged to provide the system with immutability, traceability and transparency of information. To prove the feasibility of our design, we build a Decentralized Application (DApp) operating as a marketplace for digital assets using the proposed DAM framework. Experimental results indicate that the framework is more cost-effective than existing platforms, while advanced security features are integrated and automation is maximized.**

***Index Terms*—Blockchain, Digital Asset Management, Metaverse, InterPlanetary File System, Decentralized Application.**

I. Introduction

A. Background

With the rapid development of digital platforms such as social networking, e-commerce, and especially the metaverse, creation of digital assets like photos, videos, music or metaverse items is no more limited to professionals and experts. However, digital content may lose its value overtime due to its availability and accessibility in the digital space. If a digital asset can be accessed widely and uncontrollably on the Internet, the demand for it would decrease rapidly, causing it worthless and leading to huge monetary loss for the creators.

Various solutions have been proposed to protect digital assets from illegal access, ownership, and copyright issues [1]. For example, Non-fungible Token (NFT) based on blockchain technology has emerged as an effective mechanism to secure the ownership of digital assets [2]. A digital asset can be tokenized as NFT and stored on blockchain so that one can prove the unique ownership of the asset even when the source data of the asset can be accessed by the others. However, NFT is only suitable for certain types of digital assets whose value mostly depends on their uniqueness. On the other hand, access-based digital assets should be distributed to multiple customers so that creators can take advantages of selling the content license. In this case, uniqueness is not an important factor since these digital contents can be owned by multiple people. These access-based digital assets are usually distributed through third-party authorities in traditional platforms, where the source files are stored in a centralized *Distribution channel* and license is provided through a *License broker* [3]. However, a substantial portion of profits must be shared with these intermediaries. If they act dishonestly such as leaking data to the public or not providing license after the license fee has been paid, the benefits of participants might be threatened severely. Moreover, their associated data could be leaked or altered by attackers when it is stored in centralized databases or during its transmission among different entries.

In this paper, we propose a secure Digital Asset Management (DAM) framework utilizing smart contract, InterPlanetary File System[1] (IPFS) and multi-layer encryption mechanisms to overcome all the mentioned issues. Our design eliminates the role of third parties so that all decisions are made by smart contracts to ensure the fairness for all involved stakeholders. With multi-layer encryption methods, the framework can resist data leakage even if attackers can eavesdrop all information on the transmission. Our framework also provides atomic transaction, which means that a transaction associated with the purchase of digital assets will be either successful or canceled (if the data are altered by attackers). No trust assumption among participants is required in our design, while all trading information is public and transparent on the blockchain.

B. Related Works and Motivations

Prior to our work, there have been several studies investigating the applications of blockchain technology for DAM system. The authors in [4] proposed a proof of delivery (PoD) scheme for digital assets using smart contracts. In that platform, a smart contract is deployed to be the intermediary between the asset owner and customers. When a customer is not satisfied or cannot download the files, he can report it to a third-party arbitrator. There could be a problem when the arbitrator intentionally makes wrong decision, or gets hacked, or simply is not online. Another blockchain-based digital rights management system is introduced in [5] to protect digital content. However, source files are stored in centralized cloud so it could be lost or altered by attackers. Furthermore, although it allows customers to report mismatched data, the design does not take into account the situation in which customers

T. V. Truong and L. B. Le are with INRS-EMT, University of Québec, Montréal, QC H5A 1K6, Canada (email: tuan.vu.truong@inrs.ca, long.le@inrs.ca).

[1]https://docs.ipfs.tech/

intentionally claim that they obtained invalid content for a refund, while they have actually received the correct source files. There have been also several frameworks combining blockchain and Attribute-based Acess Control (ABAC) to control data access [6]–[8]. However, none of them provide possible solutions for data leakage and data alteration. This work aims to fill these gaps in the existing literature.

The rest of this paper is organized as follows. In Section II, the DAM framework is proposed with a complete Decentralized Application (DApp) operating on top of the Ethereum blockchain as a marketplace. Section III provides analyses on performance and security features of the design. Finally, the paper is concluded with Section IV.

II. Proposed DAM Framework

In this section, we describe our smart contract-based solution in digital asset management. The overall architecture is presented with a complete workflow for different scenarios. We also provide a marketplace DApp to realize our framework with full functions which are described in the architecture.

A. Preliminary

Our framework utilizes smart contract, IFPS, Elliptic Curve Digital Signature Algorithm[2] (ECDSA) and AES-256[3] encryption to resist a wide range of attack scenarios.

1) Blockchain and Smart Contract: Blockchain technology is a chain of blocks linked together, where each block contains a certain amount of data in a decentralized manner. If those data are financial transactions such as sending tokens from one user to another node, the blockchain can be considered as a distributed ledger. Moreover, it can also store programs that run when predetermined conditions are met, which are called smart contracts. Smart contracts are usually used to automate the decision making process, thereby getting rid of intermediary's involvement. In our framework, smart contracts are deployed on the Ethereum blockchain[4].

2) IPFS: IPFS is a peer-to-peer network for storing and sharing data[5]. It is a decentralized environment where each file is divided into smaller chunks of data, and then stored by different nodes. Each file is identified by its hash, which is called Content Identifier (CID). When a user wants to download a file from IPFS, he must ask other nodes for all necessary chunks based on the file's CID. With this mechanism, IPFS ensures that all the files cannot be altered, since it will lead to the modification of the corresponding CIDs. Moreover, IPFS provides data redundancy, thus guaranteeing the availability of data as it cannot be taken down by hackers.

B. Overall Architecture

Fig. 1 illustrates the proposed DAM architecture, which consists of the following entities:

- **Content Owner:** A party who owns the digital asset and would like to sell it to customers.
- **Customer:** An entity who is interested in the digital asset. They can get the asset by sending request to the DAM smart contract.
- **DAM Smart Contract:** The smart contract regulating the operation of the framework. It can receive requests from customers, receive funds from other parties and decide whether to grant access for any eligible customer.
- **IPFS:** A decentralized storage environment for storing source files of digital assets. Customers can easily download the files from IPFS if they are granted access.

In this framework, it is assumed that all participants possess Ethereum addresses that could connect to the blockchain and communicate with the DAM smart contract. Each DAM smart contract accounts for only one particular digital asset. The workflow of the proposed platform consists of the following stages:

- **Stage 1:** The customer firstly sends a register request to the smart contract associating to the asset. The register request is actually a transaction calling to a function of the contract with a predefined license fee.
- **Stage 2:** After the register request is received by the smart contract, a new event is emitted to the content owner. If the owner decides to grant access to the customer, he extracts the customer's public key and prepares the asset's source files for further stages.
- **Stage 3:** In this stage, the content owner uses provided tools to encrypt the source file with a generated *symmectric key,* calculate hash of the encrypted file and generate a *signed key*, which can be used by only the customer who successfully buys the asset to decrypt the file. This process is presented in detail in Section II-C.
- **Stage 4:** The encrypted file from stage 3 is then uploaded to IPFS. As a result, the content owner obtains the IPFS link/URL for that file (derived from the IPFS ID).
- **Stage 5:** The content owner sends the signed key, IPFS link and the encrypted file's hash to the DAM smart contract according to the customer's address. At this point, the IPFS link can be accessed only by the associated customer, while the signed key and file hash cannot be accessed by anyone.
- **Stage 6 and 7:** The customer receives the IPFS link and uses it to download the encrypted file from IPFS.
- **Stage 8:** The customer calculates the hash of the encrypted file downloaded from IPFS. Then, he calls the smart contract function *Compare Hashes*, passing the calculated hash as an argument.
- **Stage 9:** Depending on the result of the Compare Hashes function in the previous stage, there are two scenario. If the two hashes match with each other, the smart contract will send the signed key to the customer (9.1). Otherwise, the contract is canceled and the funded license fee is sent back to the customer (9.2).
- **Stage 10 and 11:** If 9.1 occurs, the smart contract

[2]https://csrc.nist.gov/glossary/term/ecdsa
[3]https://www.nist.gov/publications/advanced-encryption-standard-aes
[4]https://ethereum.org/en/
[5]https://ipfs.tech/

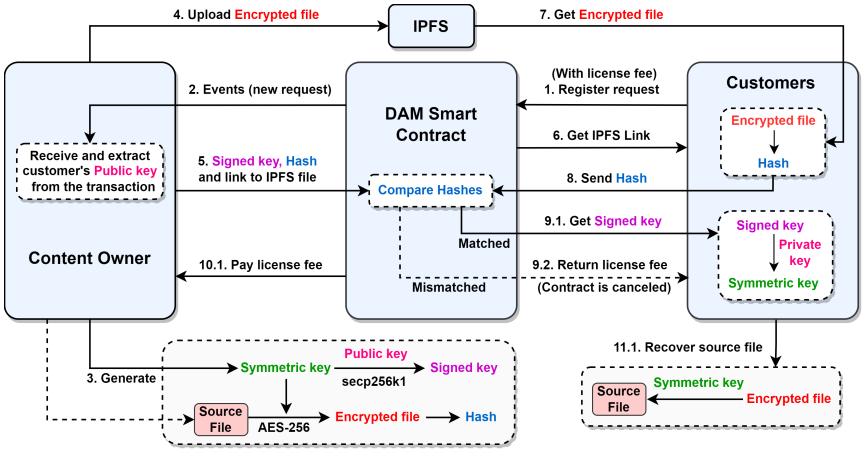

Fig. 1. The overall architecture of the proposed DAM Framework

finishes the transaction by sending license fee to the content owner. Finally, the customer can use the signed key combining with his own private key to decrypt the encrypted file, thereby obtaining the source file.

In addition, customers can cancel their request at anytime as long as they have not been granted access to the digital asset (i.e., before the stage 5). In this case, only the customer incurs a small gas fee on the transaction at stage 1. However, customers still can cancel their request after stage 5 by intentionally sending a wrong file hash to the smart contract to compare in stage 8. In this case, both the content owner and the customer lose a minor gas fee interacting with the smart contract in previous stages. To prevent attackers from draining the owner's account, each customer address can only fail at most once when comparing hashes. Otherwise, the smart contract will ban the suspicious address.

While each digital asset is managed by one smart contract, multiple customers could request for the asset at the same time. To prevent single source of truth (SSOT), a new symmetric key associated with a new IPFS link is generated for each customer. Since we use IPFS instead of centralized storage environments, storage capacity is almost unlimited and should not be considered as a significant issue.

This process ensures that there are only two possible circumstances; either the customer receives the valid source file and license fee is sent to the content owner, or the transaction is canceled without any significant financial loss. Furthermore, all transactions are submitted to the blockchain, therefore they are immutable and transparent to all participants.

C. Key Generation and Encryption Unit

This is an important process to ensure the security of the entire system. Inputs of this process include the source file of the digital asset, and the public key of the customer who requested the asset. Firstly, a 256-bit symmetric key is generated randomly by the key generation unit. Then, the following two sub-tasks are performed:

- **Source File Encryption:** The symmetric key is used to encrypt the source file using AES-256 algorithm. The encrypted file is then hashed to obtain the file hash for later usage.
- **Creating Signed Key:** Public key of the customer is extracted from his transaction. It is used to sign the symmetric key, thereby acquiring the signed key. Actually, this is an asymmetric encryption process using ECDSA (secp256k1) encryption.

As a result, the above process outputs the signed key according to the customer's public key, the encrypted file and its hash. The file hash is used to compare with the hash computed by the customer to ensure that the encrypted file is not altered during transmission or in the storage environment. Whenever the customer receives the signed key, he and only he can decrypt it to obtain the original symmetric key by using his private key.

TABLE I
MAIN FUNCTIONS OF THE DAM SMART CONTRACT.

Function	Caller	Gas	Description
Update price	Owner	Yes	Update new price for the asset
Register request	Customer	Yes	Request to buy the asset
Cancel request	Customer	Yes	Stop buying the asset
Grant access	Owner	Yes	Accept the request (Stage 5)
Get IPFS URL	Customer	No	Get IPFS url from the contract
Compare hash	Customer	Yes	Compare two hashes (Stage 8)
Get signed key	Customer	No	Get the key if hashes match
Withdraw fund	All	Yes	Withdraw available tokens
Get price	All	No	Get current price of the asset

D. Smart Contract and Marketplace Implementation

The DAM smart contract is written in Solidity and implemented on Ethereum blockchain. Our source code[6] for the DAM smart contract is available on Github. Main functions of the smart contract are shown in Table I.

Furthermore, we created a marketplace utilizing the DAM smart contract to prove its feasibility in practice. This is a complete DApp connecting to the blockchain, IPFS and provides users with a user-friendly interface (illustrated partly

[6]https://github.com/tuanvu171/Digital-Asset-Management

in Fig. 2). We also publish the source code for our DAM Marketplace on Github[7].

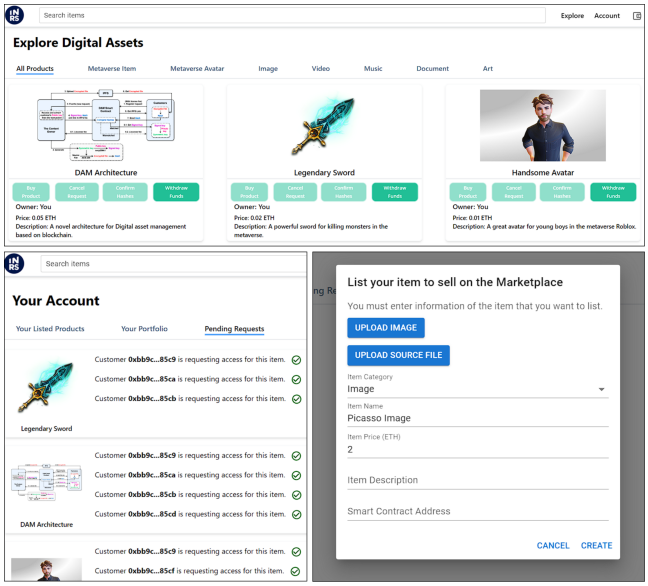

Fig. 2. The DApp marketplace operating on top of our proposed framework.

III. Performance and Security Analysis

A. Security Analysis

In this section, we analyse a wide range of scenarios attacking different components of the system, thus showing how our framework can resist these attacks. The only assumption that our system needs is that attackers cannot steal the private key from customers, which is an obvious condition for all blockchain-based systems.

1) Data Leakage: In general, an attacker could exploit the following components of the DAM system to steal data.

- **Attacking the storage environment:** In DAM systems using centralized databases, attackers can hack these databases to steal all digital assets. On the other hand, our framework uses IPFS, a decentralized file system, to store source files of digital assets. Only those who have the IPFS ID can download the corresponding files.
- **Communication between users and storage environment:** It is possible that hackers eavesdrop data when the content owner uploads the file to IPFS, or when customers download the file from IPFS (stage 4 and 7 in Fig. 1). However, the file has been encrypted using a randomly generated symmetric key, thus the attackers still cannot access the original file.
- **Communication between users and the smart contract:** Another scenario of data leakage is that attackers eavesdrop both the IPFS URL and the signed key when they are sent to the smart contract in stage 5, or when they are downloaded by customers in stage 6 and 9. With the IPFS URL, attackers can directly download the file from IPFS. However, only the steal signed key is not enough to decrypt the file. Instead, it needs the symmetric key, which can be obtained by only the customer with his private key.

Our framework is designed to withstand all scenarios leading to data leakage. Even when attackers could dominate the system and steal all data during the transmission, they still cannot obtain the source file of digital assets.

2) Data Alteration: As we use IPFS for data storage, it is impossible to alter content storing in this decentralized environment thanks to its mechanism [9]. However, attackers can interfere in the transmission of data while it is uploaded or downloaded by participants, thus altering the file intentionally. In our framework, this issue is prevented by comparing hashes as presented in stage 8 of the workflow. If the downloaded file is altered, the hash calculated by the customer will not match with the hash stored in the smart contract. As a result, the transaction is canceled and license fee is paid back to the customer. This process is completely reliable since it is carried out by the smart contract instead of any third-party authority. In other words, it offers atomic transaction, meaning that the final payment will be either successful or canceled without financial loss.

3) Participant Dishonesty: Our framework can resist participant fraud without the intervention of third-party authorities.

- **Dishonest Content Owner:** In traditional platforms, the content owner may not provide the digital asset's license or might send incorrect files after receiving license fee. In our framework, license fee is only paid after the smart contract confirms that the customer has received the correct source file. In all other cases, transactions are canceled.
- **Dishonest Customer:** It is possible that customers already received the correct file but they try to avoid paying license fee by claiming that the file is invalid or they cannot download it. In our system, customers can only download the encrypted file before the payment is settled. Without confirming receipt of the correct content, customers gain no information related to the digital asset.

Thanks to the above incentive mechanisms, our framework does not require trust assumptions among participants in the network. If a user acts dishonestly, he will lose a certain gas fee, while the system is finalized safely.

4) Smart Contract Attacks: It is necessary to take into account attacks related to smart contract vulnerabilities [10]. In this paper, we use the tool Oyente published by Luu *et al.* [11] to analyse possible financial vulnerabilities such as Re-Entrancy Attack, Callstack Depth Attack, Transaction Ordering Dependence (TOD) and Parity Multisig Bug.

Fig. 3 shows the security analyses on our DAM smart contract. The results indicate that our smart contract does not suffer from any security vulnerabilities, since all types of attack are marked "False". Therefore, it can be argued that our framework is safe from all these types of smart contract-related attacks.

[7]https://github.com/tuanvu171/DAM-Martketplace

```
INFO:root:contract DigitalAssetContract.sol:DigitalAssetContract:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                    31.6%
INFO:symExec:     Integer Underflow:                    False
INFO:symExec:     Integer Overflow:                     False
INFO:symExec:     Parity Multisig Bug 2:                False
INFO:symExec:     Callstack Depth Attack Vulnerability: False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                 False
INFO:symExec:     Re-Entrancy Vulnerability:            False
INFO:symExec:   ====== Analysis Completed ======
```

Fig. 3. Smart contract security analysis for different attack types.

5) Other Beneficial Features: Since our framework operates on top of the Ethereum blockchain, it inherits most of the key features of blockchain technology as following:

- **Eliminate Third-party-related Issues:** In traditional platforms, third-party authorities are often involved in different components of the system such as distribution channels, licensing brokers or centralized databases. If they are hacked or they act maliciously, the entire system will be completely dominated. In contrast, our framework does not depend on third parties. Therefore, it eliminates all issues related to intermediaries.
- **Minimize Communication:** In our platform, there is no communication between participants. Instead, they mostly interact with the DAM smart contract using the DApp.
- **Lower Cost:** No commission is required in our platform. The only costs users must pay are gas fees for transactions on the blockchain network. They will be presented in detail in Section III-B1.
- **Transparent Information:** Another advantage of our blockchain-based framework is that all information related to trading digital assets are public, transparent and immutable since they are stored in the blockchain.

B. Performance Analysis

In this section, we provide analyses on the performance of proposed framework. Firstly, we analyse gas consumption and operation cost of the system in different scenarios. Then, we estimate the speed and latency of the framework based on the figures of blockchain and IPFS.

1) Gas Estimation: Table II shows the gas consumption of each contract function in the framework. Contract deployment only runs once when the smart contract is initialized. Functions which are more complex (e.g., containing intensive computation) would consume more gas. Other functions which only read information from the blockchain and do not change the blockchain state consume no gas. On the other hand, gas fee (in ETH) is mainly determined by the supply and demand among miners, while the price of ETH is decided by the market. During the analysis on September 2022, 1 gas is approximately 8 gwei (1 gwei = 10^{-9} ETH), while 1 ETH is around $1,327 US.

Fig. 4 illustrates gas consumption of each participant in different scenarios of a full operating cycle (i.e., from the time the customer submit request until the access is granted successfully or the contract is canceled). The costs are calculated based on which function must be called during the process.

TABLE II
GAS CONSUMPTION OF SMART CONTRACT FUNCTIONS.

Contract function	Gas Used	Ether	USD
Contract Deployment	926655	0.00927	12.17
Register Request	59540	0.00048	0.63
Cancel Request	51836	0.00041	0.55
Grant Access	101779	0.00081	1.08
Compare Hashes (Failed)	55868	0.00045	0.6
Compare Hashes (Success)	78591	0.00063	0.84
Update Price	28794	0.00023	0.31
Withdraw Fund	29046	0.00023	0.31

TABLE III
GAS COMPARISON BETWEEN DIFFERENT FRAMEWORKS FOR EACH SUCCESSFUL OPERATING CYCLE.

Framework for Data Access Control	Cost per Successful Operating Cycle	
	Total Gas Used	Cost in USD
[8]	304,243	3.22
[7]	5,006,610	53.01
[6]	310,136	3.28
Proposed Framework	239,910	2.54

In the successful case, customers and the content owner must pay about $1.46 US and $1.08 US respectively for the overall transaction.

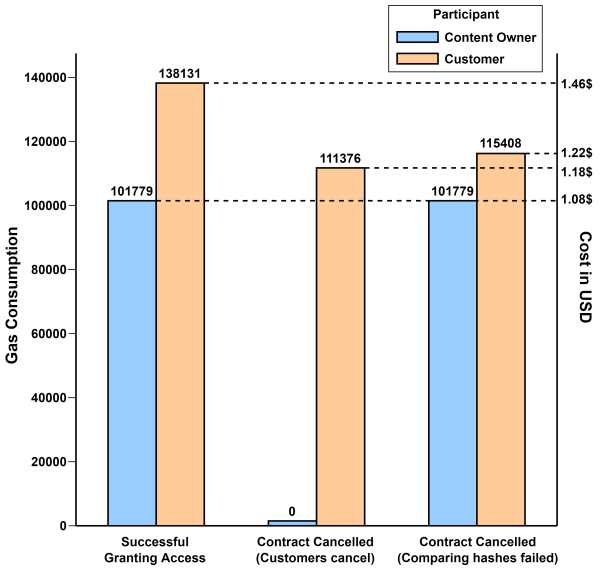

Fig. 4. Operation cost in different scenarios.

In comparison with other similar frameworks proposed in [6]–[8], the total cost of our platform is significantly lower for each successful operating cycle (showing in Table III).

2) Speed and Latency: Speed and latency of the framework depends mostly on the following factors:

- **Blockchain Platform:** Since the system is implemented on top of a blockchain, every action can only be done after a new block is mined. Block time on Ethereum blockchain is around 12 seconds. However, to ensure transactions are not reverted, we often wait for 6 to 10

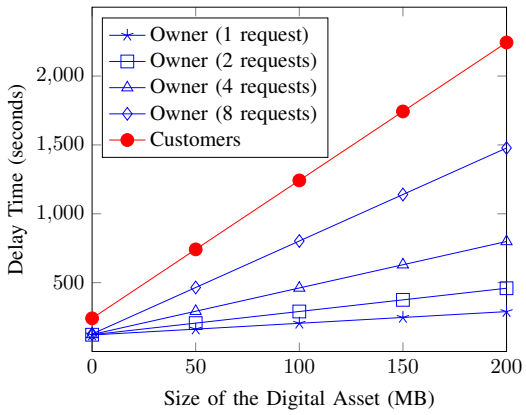

Fig. 5. Total delay time of trading digital assets for the content owner and customers according to data size and number of concurrent requests.

block times, which is about 120 seconds. Therefore, the delay causing by blockchain for each transaction can be estimated: $T_{\text{BC}} = 120s$

- **IPFS:** Actions that require data upload/download will experience higher latency depending on data size. According to [12], latency and throughput of a write operation with IPFS are around 42 ms and 1.2 MB/s respectively. Whereas, the figures for read operations are about 800 ms and 100 KB/s. As a result, the total time for uploading and dowloading data with IPFS are: $T_{\text{I-U}} = 0.042 + \frac{s}{1.2}$ and $T_{\text{I-D}} = 0.8 + \frac{s}{0.1}$ (with s is the size of the source file in MB).
- **Encyprtion and Hashing Process:** This process is to encrypt the source file of digital assets with AES-256 algorithm, and hash the files with SHA-256. With CPU frequency of 2.2 GHz, the average speed of the AES-256 and SHA-256 algorithm are about 129 MB/s and 139 MB/s respectively[8]. Thus, delay causing by AES encryption/decryption is: $T_{\text{AES}} = \frac{s}{129}$. Whereas, the figure for SHA-256 is: $T_{\text{SHA}} = \frac{s}{139}$.

All operations which only get data from the blockchain will be finished immediately without having to wait for block confirmation. Therefore, we only must calculate the total delay time for the following two actions:

- **The Content Owner:** the owner encrypts and hashes the source file, uploads it to IPFS and submits these information to the smart contract: $T_{\text{Owner}} = T_{\text{BC}} + T_{\text{AES}} + T_{\text{SHA}} + T_{\text{I-U}}$.
- **Customers:** they firstly request for the digital asset. Then they download and calculate hash for the encrypted file, call the compare hash function and then decrypt the file: $T_{\text{Customer}} = 2.T_{\text{BC}} + T_{\text{AES}} + T_{\text{SHA}} + T_{\text{I-D}}$.

Fig. 5 illustrates the total latency that the content owner and customers have to wait for each complete transaction selling/buying digital assets. Customers must wait longer for the asset since they have to call two functions of the smart contract, and download data from IPFS is significantly slower than upload. Actually, the content owner can grant access to multiple customers at the same block time. In that case, the delay time only increases corresponding to encryption and download time, while all requests are arranged into a single block of the blockchain, thus T_{BC} remains unchanged.

[8]https://cryptopp.com/benchmarks-amd64.html

IV. Conclusion

In this paper, we proposed a novel blockchain-based DAM framework using smart contract, IPFS, and encryption mechanisms to control digital asset accession in metaverse platforms. Thanks to the DAM smart contract, decentralization and fairness are ensured for all decisions, while the role of third-party arbitrator is eliminated. Besides, IPFS storage and encryption methods guarantee the integrity of data so that our design is resist to data alteration and data leakage. A complete DApp is also constructed as a marketplace for digital assets to prove the feasibility of our framework in practice. In comparison with existing platforms, experimental results show that our framework consumes significantly less resources. Although the prototype has proven the feasibility of the framework, operating on a public blockchain makes it less efficient due to the natural high latency of blockchain. Therefore, our future work is to develop a consortium blockchain which is scalable and optimized for the proposed DAM framework.

References

[1] M. Barni and F. Bartolini, *Watermarking Systems Engineering: Enabling Digital Assets Security and Other Applications*. CRC Press, 2004.
[2] U. W. Chohan, "Non-fungible tokens: Blockchains, scarcity, and value," *Critical Blockchain Research Initiative (CBRI) Working Papers*, 2021.
[3] W. Ku and C.-H. Chi, "Survey on the technological aspects of digital rights management," in *Proc. International Conference on Information Security (ISC)*, 2004, pp. 391–403.
[4] H. R. Hasan and K. Salah, "Proof of delivery of digital assets using blockchain and smart contracts," *IEEE Access*, vol. 6, pp. 65 439–65 448, 2018.
[5] A. Garba, A. D. Dwivedi, M. Kamal, G. Srivastava, M. Tariq, M. A. Hasan, and Z. Chen, "A digital rights management system based on a scalable blockchain," *Peer-to-Peer Networking and Applications*, vol. 14, no. 5, pp. 2665–2680, 2021.
[6] Y. Zhang, M. Yutaka, M. Sasabe, and S. Kasahara, "Attribute-based access control for smart cities: A smart-contract-driven framework," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6372–6384, 2020.
[7] H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proc. International Conference on Blockchain Technology*, 2019, pp. 6–11.
[8] J. P. Cruz, Y. Kaji, and N. Yanai, "Rbac-sc: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12 240–12 251, 2018.
[9] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[10] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Proc. International Conference on Principles of Security and Trust*, 2017, pp. 164–186.
[11] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 254–269.
[12] J. Shen, Y. Li, Y. Zhou, and X. Wang, "Understanding i/o performance of ipfs storage: a client's perspective," in *Proc. IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, pp. 1–10.