electronics

Article

Secure Processing and Distribution of Data Managed on Private InterPlanetary File System Using Zero-Knowledge Proofs

Kyohei Shibano [1,*], Kensuke Ito [1], Changhee Han [2], Tsz Tat Chu [2], Wataru Ozaki [2] and Gento Mogi [1]

1 Department of Technology Management for Innovation, School of Engineering, The University of Tokyo, Tokyo 113-8656, Japan
2 Callisto Inc., Tokyo 171-0022, Japan
* Correspondence: shibano@tmi.t.u-tokyo.ac.jp

Abstract: In this study, a new data-sharing method is proposed that uses a private InterPlanetary File System—a decentralized storage system operated within a closed network—to distribute data to external entities while making its authenticity verifiable. Among the two operational modes of IPFS, public and private, this study focuses on the method for using private IPFS. Private IPFS is not open to the general public; although it poses a risk of data tampering when distributing data to external parties, the proposed method ensures the authenticity of the received data. In particular, this method applies a type of zero-knowledge proof, namely, the Groth16 protocol of zk-SNARKs, to ensure that the data corresponds to the content identifier in a private IPFS. Moreover, the recipient's name is embedded into the distributed data to prevent unauthorized secondary distribution. Experiments confirmed the effectiveness of the proposed method for an image data size of up to 120 × 120 pixels. In future studies, the proposed method will be applied to larger and more diverse data types.

Keywords: IPFS; zero-knowledge proof; circom; zk-SNARKs; private IPFS; data distribution; data processing; data security

Citation: Shibano, K.; Ito, K.; Han, C.; Chu, T.T.; Ozaki, W.; Mogi, G. Secure Processing and Distribution of Data Managed on Private InterPlanetary File System Using Zero-Knowledge Proofs. *Electronics* **2024**, *13*, 3025. https://doi.org/10.3390/electronics13153025

Academic Editor: Aryya Gangopadhyay

Received: 13 June 2024
Revised: 14 July 2024
Accepted: 20 July 2024
Published: 31 July 2024

1. Introduction

Decentralized systems are robust because they lack a single point of failure; therefore, they are widely applied across enterprise sectors including cryptocurrency, supply chain management, financial services, and digital identity. To store large-sized data such as images, these systems require storage functions that are inherently decentralized. Blockchain, commonly used in conjunction, typically handles smaller data sizes such as transaction histories and operates as a ledger database. The InterPlanetary File System (IPFS) is a prominent decentralized storage system that stores data across multiple nodes to enhance data availability. The IPFS has two variants: public IPFS, wherein the data can be stored by any user with unrestricted access, and private IPFS, wherein a closed network accessible only within specific organizations or groups is established, offering enhanced privacy and security.

When storing data in IPFS, understanding the differences between public IPFS and private IPFS is crucial. Public IPFS allows anyone to access data, while private IPFS is accessible only within specific organizations or groups, enhancing privacy and security. When storing sensitive information, such as confidential data, in public IPFS, applying an appropriate encryption scheme is vital to ensure data protection. By contrast, private IPFS provides higher security for data storage, because it is accessible only within a closed network.

Particularly for organizations such as corporations or healthcare institutions, storing data in public IPFS, despite using strong encryption technologies, carries inherent risks. Moreover, the potential for data leaks due to operational errors exists persistently in such cases; although data is encrypted, it is exposed to the world, rendering it vulnerable to brute force attacks and other security threats.

Regarding accessibility, public IPFS allows general users to directly access and retrieve data. However, in private IPFS, data must be received from members of the organization or group constituting the network. During this process, if data is tampered, then users may unable to detect it. Therefore, trusting the intermediaries responsible for handling data transfer in such cases becomes mandatory. To address the aforementioned trust issue, a new method is proposed herein for distributing data stored in a private IPFS to external entities while making its authenticity verifiable. The Groth16 [1] protocol of zk-SNARKs, a type of ZKP, is applied to data stored in a private IPFS to ensure the authenticity of the data. Moreover, the recipient's information is embedded into the distributed data to prevent unauthorized secondary distribution. The proposed method of data sharing is important because it is tailored to the private IPFS case.

The differences in several aspects, including security and accessibility, when storing data in public IPFS and private IPFS within the enterprise domain are summarized in Table 1. This study proposes solutions to the threats associated with private IPFS.

Table 1. Comparison between public and private IPFS in the enterprise domain.

	Public IPFS	**Private IPFS**
Trust Model	Trustless	Requires trust in the operating group
Access Restrictions	Accessible by anyone	Accessible only within the operating group
Data Leakage Risk	Constant risk of leakage due to user error	Low risk of leakage within a closed network
Handling of Confidential Information	Requires proper encryption	Data stored in IPFS does not require high-level encryption itself; there is a trust point when passing data to users
Brute Force Attack Risk	Always present	Low
Data Retrieval Method	Direct access by users	Data received from members of the organization or group
Threats	Requires encryption that prevents decryption by unauthorized users	There is a risk of tampering when transferring data to users

The remainder of this paper is organized as follows. Section 2 presents related prior research. Section 3 outlines the fundamental technologies, i.e., ZKP and zk-SNARKs. Section 4 describes the structure of the proposed method, while Section 5 outlines the potential applications of this method. Section 6 presents the implementation of this method, while Section 7 discusses the experiments performed to verify the effectiveness of the implementation. Section 8 presents a discussion of the experimental results, while Section 9 presents the conclusions of the paper and an outline of future challenges.

2. Related Studies

Existing decentralized systems use IPFS, particularly in combination with blockchain technology. Kumar et al. [2] proposed a method for securely managing medical data by integrating IPFS with a blockchain. Azbeg et al. [3] specifically suggested a system that managed and stored medical data using private IPFS and a permissioned blockchain by employing proxy re-encryption to ensure secure decryption by designated doctors. When a physician receives some patient's data, he/she obtains the re-encrypted data via a hospital. Hossan et al. [4] also proposed a system to securely record information for ride-sharing services using IPFS and a private blockchain.

Focusing on controlling the distribution of data managed by IPFS, Lin et al. [5] proposed a system for protecting private data using improved IPFS combined with a blockchain. This system recorded file metadata and accessed permissions on the blockchain, enabling users to control file sharing. Moreover, the system implemented efficient management features using smart contracts, thereby enhancing data security and management flexibility. Battah et al. [6] developed a system that used multiparty authentication (MPA), proxy re-encryption, and smart contracts on a blockchain for decentralized access control of encrypted data stored in IPFS. Huang et al. [7] introduced a trusted IPFS proxy to realize access control and group key management for encrypted data stored in IPFS. Sun et al. [8] proposed a system that allowed only individuals with appropriate attributes to decrypt encrypted data stored in IPFS using a ciphertext policy attribute–based encryption system, facilitating efficient medical information management. Kang et al. [9] enabled the distribution of data managed using private IPFS and a private blockchain to external users using named data network (NDN). Furthermore, Uddin et al. [10] proposed a file-sharing system that used IPFS and public key infrastructure (PKI) technology without requiring a trusted third party.

Several studies have used ZKP for data distribution. For instance, Li et al. [11] proposed a privacy-preserving traffic management system that combined noninteractive zero-knowledge range proofs with a blockchain. A prototype using Hyperledger Fabric and Hyperledger Ursa met the data privacy requirements for real-time traffic management.

This study proposes a method for appropriately processing and distributing data managed within private IPFS to users outside the network, thereby offering a different approach than those proposed in previous studies. Some studies have adopted proxy re-encryption as an appropriate method for data storage and distribution in IPFS [3,6]. Using this method, distributed data can be re-encrypted to be decrypted with the recipient's private key. Moreover, when storing data in IPFS, recording the hash value of the pre-encrypted data on the blockchain allows recipients to verify the correctness of their received data after decryption. However, this method cannot handle cases where data is processed, such as embedding the recipient's name into the decrypted data, as in this study.

3. Zero-Knowledge Proof

ZKP is a cryptographic protocol that allows a prover to prove the validity of a proposition to a verifier without disclosing any additional information other than the validity of the proposition. The proposition of this study is that the data provided to an external entity is generated based on a given CID. Our goal is to allow a member of private IPFS (prover) to prove this proposition to an external entity (verifier as the recipient of the data) without disclosing any other important information (such as IPFS access rights and encryption keys).

ZKP, specifically the Groth16 protocol of zk-SNARKs used in this study, begins with a trusted setup where both parties establish public parameters that are crucial for the secure generation and verification of proofs. In the ZKP scheme, we first generate a circuit that describes the process for which a proof is intended. The circuit includes the conditions to be verified such as the existence of a CID. Through the ZKP scheme, cryptographic keys—specifically a proving key and a verification key—are created. These keys are crucial for creating a proof for the circuit and its verification. Using the proving key and input data, the prover generates a proof that reveals the validity of the output data against the conditions specified in the circuit. The verifier then uses the verification key to check the proof and the output data. If the proof is valid, this confirms the integrity of data without exposing any underlying information.

In this study, Groth16 processing is performed using circom [12,13] and snarkjs [14]. The process flow is summarized in Figure 1.

zk-SNARKs is employed owing to its noninteractive nature and efficiency, which are particularly advantageous for systems employing smart contracts owing to low computational costs for verifying the proof. Furthermore, zk-SNARKs is known for its high computational requirements and the need for advanced PC specifications. For instance, in one of the representative applications of zk-SNARKs, Zcash [15], proof generation process takes over half a minute for a single anonymous transaction [16].

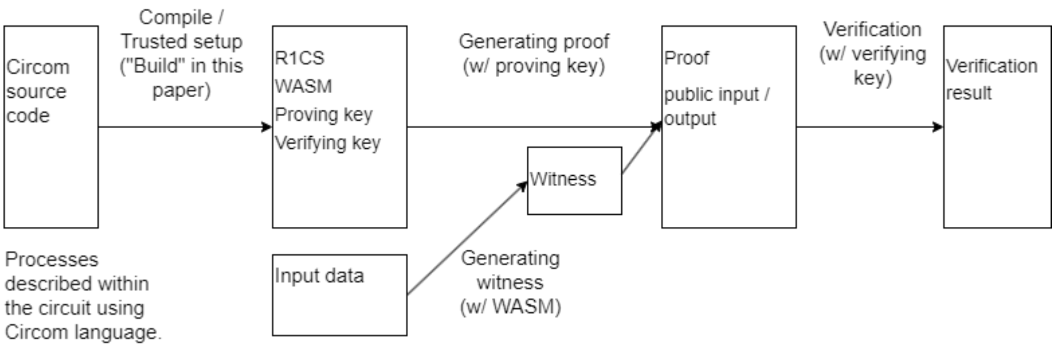
Circom source code
Compile / Trusted setup ("Build" in this paper)
R1CS
WASM
Proving key
Verifying key
Generating proof (w/ proving key)
Proof
public input / output
Verification (w/ verifying key)
Verification result
Witness
Input data
Generating witness (w/ WASM)
Processes described within the circuit using Circom language.

Figure 1. Zero-knowledge proof using circom.

4. Proposed System

Figure 2 presents an overview of the proposed system.

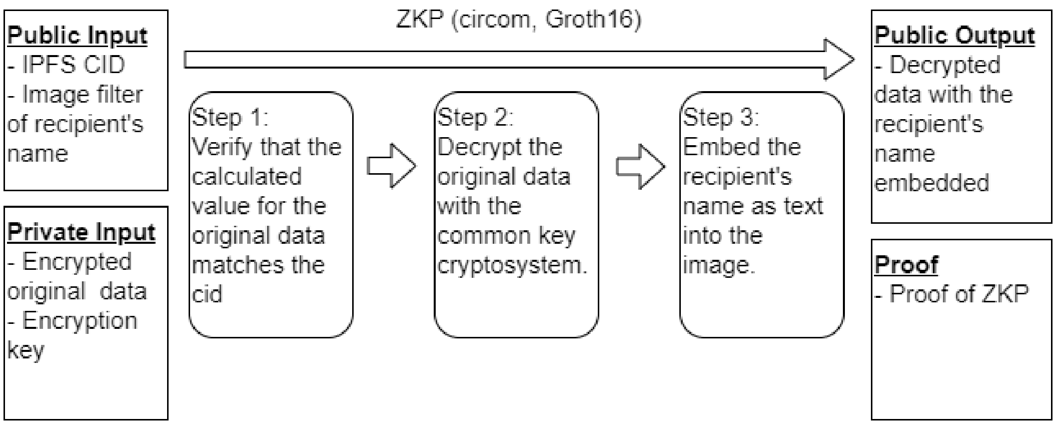
ZKP (circom, Groth16)
Public Input
- IPFS CID
- Image filter of recipient's name
Private Input
- Encrypted original data
- Encryption key
Step 1: Verify that the calculated value for the original data matches the cid
Step 2: Decrypt the original data with the common key cryptosystem.
Step 3: Embed the recipient's name as text into the image.
Public Output
- Decrypted data with the recipient's name embedded
Proof
- Proof of ZKP

Figure 2. Process flow of ZKP: an example of image data processing.

Herein, we make the following assumptions:

- Private IPFS is operated by a limited number of members.
- Data are stored in IPFS in an encrypted format using symmetric-key cryptography.
- The encryption key is exclusively held by an individual among the members, i.e., an administrator.

The proposed system facilitates the creation of a ZKP proof through the circuit by the encryption key-holding member (equivalent to an administrator). The inputs and outputs (other than proofs) of this process are as follows:

- **Public Input:**
  - CID of the original encrypted data;
  - A filter for embedding the recipient's name in the data;
- **Private Input:**
  - Encrypted original data;
  - Encryption key;
- **Output:**
  - Decrypted data with the recipient's name embedded.

Note that the ZK-optimized implementation (Section 6.1) adds more information to the public input. In particular, the internal process of ZKP involves the following steps:

1. Calculating the CID of the original encrypted data to verify a match with the entered CID.

2. Decrypting the original data using a symmetric-key cryptography.
3. Embedding the recipient's name into the decrypted image raw data based on the filter provided in the public input.

In the proposed system, we do not solely focus on image data but use them as example data to verify the applicability of the proposed scheme. Filters are used to improve the efficiency of processing inside the circuit. As embedding name data inside the circuit is computationally intensive, a considerable portion of the image processing is performed outside the circuit in advance and a filter is created. Using public input and proof, the recipient can verify that the decrypted data (i.e., output) with their name embedded are generated from the original data (contained in the private input) managed with the CID. The recipient can check with at least one member of the network to confirm the existence of the CID in private IPFS.

In summary, the aforementioned process enables the recipient to verify the received data by performing the following tasks:

- verify that the received data were generated from the data managed with the CID of private IPFS,
- confirm using the proof that the entire process was correctly conducted without directly knowing the encrypted data or the encryption key, and
- verify that the CID exists specifically within the private IPFS by asking at least one network member.

The novelty of the proposed system is that it allows data authenticity verification by trusting at least one member of the network even if the recipient do not control the encryption key. (It is natural for the recipient to trust at least one member of a particular multimember system. If none of the members can be trusted, then there will be a marginal incentive to receive data managed by that network).

5. Potential Applications

In the medical industry, patient diagnosis data are managed across multiple medical institutions. Using the proposed system, patients can verify whether the data they receive are indeed managed in private IPFS to ensure the authenticity. An all-in-one platform is also proposed herein for the research and development of machine learning with medical images [17]. On this platform, anonymized medical images are managed in private IPFS operated by a group of medical institutions. The system allows machine learning researchers, who are external to the network, to verify whether the image data are indeed managed in the private IPFS. Moreover, by embedding the information about machine learning researchers in the image data, medical institutions can mitigate the risk of secondary distribution.

If the application is not limited to the embedding of recipient's name, the potential applications of the proposed system can be further expanded. For instance, consider a scenario where a specific company establishes private IPFS for sharing confidential documents among its group companies. If employee data are included, then concealing private data and distributing them to external entities allows these entities to confirm the association of employees with the company while ensuring that their privacy is protected. Furthermore, suppose a university has set up private IPFS to allow only academic staff access to student performance data. In this case, students can verify that their performance data received are genuinely managed in the private IPFS.

Thus, the proposed system supports a hybrid case—distributing internal data to specific external entities as necessary—prevalent in real-world settings.

6. Implementation

The proposed system was implemented to process image data using circom, a renowned tool specialized for constructing zk-SNARKs circuits. Circom enables the description of computational processes within a circuit using its unique language, and the executable file generated after compilation can be invoked via the JavaScript library, snarkjs. This arrangement allows describing circuit processes in circom, and external processing and circuit correctness testing are performed using JavaScript. For the zk-SNARKs scheme, Groth16 was used; it is known for its relatively faster execution speed than other zk-SNARKs scheme.

In particular, we worked on two types of implementations for image data: a standard implementation using general cryptographic techniques and a ZK-optimized implementation using ZK-friendly cryptographic techniques to reduce the computation time of the circuit. These implementations were used for comparing the required computation times. ZKP circuits require considerably large computation time, even for calculations that can be easily handled by computer software (this is particularly noticeable when dealing with image data). Therefore, computational efficiency is crucial for practical use.

6.1. Standard Implementation

Section 4 describes the data input into the circuit. For simplifying the in-circuit processing, the original encrypted data were formatted as bitmap image data compliant with OS/2 standards. The first 54 bytes of the image data store information such as the width, height, and color depth of images [18]. The color depth is 8 bits and each color component in RGB is allocated one byte, resulting in a representation of 3 bytes per pixel.

Initially, the system checks whether the encrypted data, entered as a private input, matches the CID provided as a public input. If they do not match, the system signifies an error and the image data outputted as the output is a byte sequence where all values are 0x00. CID serves as crucial mechanism for uniquely identifying files and efficiently retrieving data from IPFS. CID has two versions: V0 and V1 [19]. Herein, the more flexible version CID V1 was used. CID includes a hash of the respective data, ensuring different data will have different CIDs. Typically, CID V1 is calculated using the SHA256 hash function, and the standard implementation uses SHA256 to compute CID.

The data structure of CID V1 is as shown in Table 2.

Table 2. CID V1 data structure.

Byte Position	Description	Value in Implementation
First byte	CID version	0x01
Second byte	multibase prefix	0x55: raw data
Third byte	Hash function identifier	0x12: SHA-256
Fourth byte	Hash length	0x20: 32 bytes
From fifth byte	Hash value	SHA-256 hash value (32 bytes)

The encoding for CID is conducted using Base32. Base32 encodes a sequence of bytes constructed based on this structure to generate CID.

Inside the circuit, the entered CID value is decoded from Base32 and the system checks whether the extracted hash value matches the SHA256 hash computed from the encrypted data.

Subsequently, the encrypted data are decrypted. AES-CTR is used as the encryption algorithm, which is a type of symmetric-key cryptography. The AES-CTR encryption and decryption in circom-chacha20 [20] was used. For decrypting AES-CTR encryption, the encryption key and nonce used during encryption are required. They are input into the circuit as a 256-bit key and a 128-bit nonce, respectively, as private inputs. Moreover, AES-CTR handles data volumes in multiples of 16. Therefore, if the length of the image data before encryption is not a multiple of 16, zeros (0x00) are added to the end of the data to align it with this requirement.

Finally, a filter is applied to the decrypted data to embed the recipient's name. Implementing text embedding directly within the circuit can substantially increase the computation load; therefore, a filter is created outside the circuit that performs a considerable portion of the image processing in advance. The font used for the text representing the recipient's name is the Misaki font [21]. The filter is then used to streamline processing inside the circuit. The filter is a list of numbers where values from 0 to 255 are used to change the color of each pixel in case it differs from that of the pixels in the original image; moreover, a value of 300 indicates the color should remain as in the original image. This filter represents the position on the image where the recipient's name should be inserted. Inside the circuit, the specified pixel colors in the decrypted BMP data are changed based on this filter.

6.2. ZK-Optimized Implementation

ZK-optimized implementation changes the hash function, encryption technology, and in-circuit processing to the standard ZK-friendly encryption implementation. This implementation enhances the computational efficiency and does not evaluate the difference in computation speeds between ZK-friendly encryption and general encryption. Therefore, in-circuit processing was also modified.

Poseidon hash [22] was used as the hash function for computing CID. Notably, using the Poseidon hash for CIDs is not officially supported; therefore, it was developed specifically for this study. Although SHA256 is commonly used in general computations, it demands considerable computation time within ZKP circuits. The Poseidon hash is implemented in circom and JavaScript (circomlib [23] and circomlibjs [24], respectively). It is computed over a finite field with a prime order and can accept up to 16 input variables. The used order is less than the maximum of 32 bytes but greater than the maximum of 31 bytes. This indicates that each of the 16 inputs must contain data not exceeding this order. In this implementation, the data targeted for hash computation are divided into 31-byte segments as input values. If the division exceeds 16 segments, the Poseidon hash is calculated for the first 16 segments. This result is added to the next 15 segments of data for a subsequent Poseidon hash input. The process is repeated until all the input data are used for hash computation. Computationally, if the final input does not complete 16 segments, the missing inputs are set to zero to ensure that the computation always involves 16 inputs.

When generating CID from the Poseidon hash value, the byte sequence should follow the CID V1 data structure and be Base32-encoded. However, to further reduce computation time, this implementation omits the Base32 encoding and directly uses the Poseidon hash value as a substitute for CID. Dividing the input data into 16 segments within the circuit is computationally intensive; therefore, this division is performed outside the circuit and given as an input. In this case, the encrypted data byte sequence and the list of values for calculating the Poseidon hash are provided as public inputs, allowing the verification that both datasets represent the same information. Recipients can confirm that the data being computed for the Poseidon hash and the data being decrypted in the circuit are identical by mutually converting and checking these two values. In this case, as users can obtain the decrypted data, a concern exists regarding password leakage through brute force attacks or other means.

For encryption technology, we adopted Poseidon encryption [25] instead of AES-CTR encryption. Poseidon encryption, implemented in circom and TypeScript (poseidon-encryption-circom2 [26]), involves receiving the public key of the recipient, generating a common key, and ensuring secure encryption and decryption by both parties. In this case, however, a common key is directly generated and used for encryption and decryption. The circuit is provided with two values representing the coordinates of an elliptical curve and a nonce value as private inputs for encryption. Moreover, the filter is implemented in the same manner as in the standard implementation.

7. Evaluation

We created a sample program based on the aforementioned implementations that uses circom to describe the circuit and uses snarkjs for executing the circuit and verifying proofs. As cryptographic libraries, circom-chacha20 [20], circomlib [23], circomlibjs [24], and poseidon-encryption-circom2 [26] were used.

The standard and ZK-optimized implementations were implemented for each circuit, and their computation times were compared during execution. White bitmap images were the target images, and the experiments were conducted using the letter "A" as the embedded character. As embedding any number of characters does not alter the processing by the filter, embedding a single character allowed for comparing the computation times. Furthermore, we varied the image sizes to measure the execution times for each circuit. The sizes used were 10 × 10, 15 × 15, 30 × 15, 30 × 30, 60 × 30, 60 × 60, 120 × 60, 120 × 120, and 180 × 120 pixels. The execution environment was Windows 11 with a Ryzen 9 3950X CPU and 128 GB RAM operating under Ubuntu 22.04 in a WSL2 environment.

Figure 3 shows an example image generated by the circuit, specifically for the 60 × 30 pixel size using the ZK-optimized implementation. The results for each image size are presented in Table 3, where nonlinear constraints indicate the number of nonlinear constraints in the circuit, build time is the time required to compile circom and output the circuit, and proof gen time is the time required to generate proofs using the circuit. As standard implementation uses AES-CTR encryption, data with 0x00 are appended at the end to ensure that the input size is a multiple of 16.

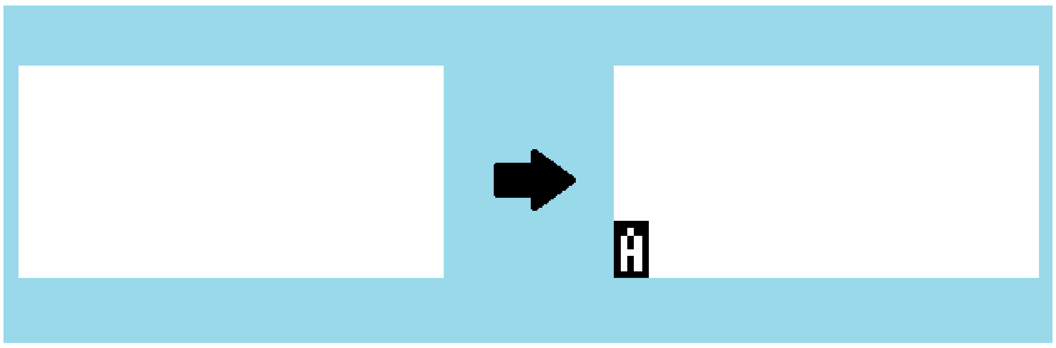

Figure 3. An image generated by the circuit for a 60 × 30 image size by ZK-optimized implementation.

Table 3. Comparison of the execution time of the circuit.

	Pixel	Image Size [Byte]	Nonlinear Constraints	Build Time [ms]	Proof Gen Time [ms]
Standard					
	10 × 10	384	558,341	668,220	14,377
	15 × 15	784	1,095,292	1,316,666	25,545
	30 × 15	1440	1,980,688	1,696,370	35,161
	30 × 30	2816	3,867,989	3,657,578	65,785
	60 × 30	5456	7,453,300	7,864,583	126,262
ZK-optimized					
	10 × 10	376	35,407	128,979	3076
	15 × 15	775	72,725	173,210	4032
	30 × 15	1435	134,663	275,630	5900
	30 × 30	2815	263,450	470,872	9733
	60 × 30	5455	509,375	850,612	17,571
	60 × 60	10,855	1,013,483	1,257,418	29,044
	120 × 120	43,255	4,036,913	6,754,928	96,580

In standard and ZK-optimized implementations for 60 × 60 pixel and 180 × 120 pixel image sizes, the system ran out of memory and the computation could not be completed. In ZK-optimized implementation, the number of nonlinear constraints was reduced to approximately one-tenth that of the standard implementation for the same image size.

This reduced the build and proof generation times. However, the maximum manageable image size was still only up to 120 × 120 pixels, which is considerably small for practical applications.

8. Discussion

Although ZK-friendly cryptographic technologies were used and in-circuit processes were optimized during ZK-optimized implementation, the maximum manageable image size was approximately 120 × 120 pixels. This limits the practical utility to considerably small image sizes. However, research aimed at enhancing the performance of ZKPs is ongoing, and future technological advancements may enable handling larger image sizes. For instance, Zhang et al. [27] achieved a tenfold acceleration of zk-SNARKs using ASICs. Ma et al. [16] similarly used a graphics processing units to accelerate the proof generation time, achieving up to 48.1 times faster performance compared with traditional methods. Moreover, methods to simplify computational processes have been proposed, such as the "folding" method. This method compresses the propositions being proved [28]. As speed enhancements are being progressively studied, memory consumption will also likely be optimized. This will potentially allow handling of larger image sizes in the future.

Furthermore, we found that our proposal method can handle data sizes approximately 10 KB. Although directly applying our proposal to realistic image data (ranging from several MBs to dozens of MBs) is challenging, splitting data into chunks by modifying the encryption and embedded strings might make the application feasible.

Moreover, our implementations requires a value based on the size of the original data to be processed (encrypted) as an argument during circuit generation. Therefore, a circuit must be generated for each data. The circuit generation time (build time) increases considerably with image data size; for instance, even in ZK-optimized implementation, generating a circuit for a 120 × 120 image size requires more than 112 min (6,754,928 ms). However, once the circuit is generated, the proof generation time under the same conditions is short, approximately 97 s (96,580 ms). In other words, once a circuit is generated, proof generation is not time intensive. This fact does not pose any practical issues in cases wherein the same image is distributed to various people.

In ZK-friendly implementations, encrypted data is inputted as a public input. Handling encryption keys for images requires careful consideration. Data managed in private IPFS are encrypted. However, if encryption keys are leaked, the encrypted data could be decrypted. Therefore, specific users managing private IPFS should become administrators to carefully manage the keys or a consortium-type blockchain could be established on the same network to set and manage access rights appropriately.

9. Conclusions

A new method was proposed herein to distribute data stored in private IPFS to external entities while making its authenticity verifiable. The method applied a type of ZKP, zk-SNARKs, to verify the CID of data and embed the recipient's name. This approach enables external entities to verify that the received data are generated from the original data in private IPFS without requiring details such as IPFS access rights and encryption keys.

A standard implementation using conventional cryptographic techniques and a ZK-optimized implementation using ZK-friendly cryptographic schemes were implemented to enhance the computational efficiency of the proposed method. Experiments with a sample program confirmed the effectiveness of the proposed method for an image data size of up to 120 × 120 pixels.

This proposed method extends the usable range of decentralized storage systems to a hybrid case—distributing internal data to specific external entities as necessary. This study paves a new way for sharing sensitive information across different sectors within and outside a group. However, for the wide practical applicability of the proposed method to larger and more diverse data types, such as images and videos, processing speed must be improved and data splitting methods must be used, which are within the scope of our future studies.

Author Contributions: Conceptualization, K.S., C.H., T.T.C. and W.O.; writing—original draft preparation, K.S.; writing—review and editing, K.S. and K.I.; supervision, G.M.; project administration, K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was conducted as a collaborative research project between the University of Tokyo and Callisto Inc., funded by Callisto Inc. This work has been supported by Endowed Chair for Blockchain Innovation and the Mohammed bin Salman Center for Future Science and Technology for Saudi-Japan Vision 2030 (MbSC2030) at The University of Tokyo.

Data Availability Statement: The source code used for the simulations is available on GitHub. https://github.com/blockchaininnovation/circom_image_processing (accessed on 21 March 2024).

Conflicts of Interest: Author Changhee Han, Tsz Tat Chu and Wataru Ozaki were employed by the company Callisto Inc. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Groth, J. On the size of pairing-based non-interactive arguments. In Proceedings of the Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, 8–12 May 2016; Proceedings, Part II 35; Springer: Berlin/Heidelberg, Germany, 2016; pp. 305–326.
2. Kumar, S.; Bharti, A.K.; Amin, R. Decentralized secure storage of medical records using Blockchain and IPFS: A comparative analysis with future directions. *Secur. Priv.* **2021**, *4*, e162. [CrossRef]
3. Azbeg, K.; Ouchetto, O.; Andaloussi, S.J. BlockMedCare: A healthcare system based on IoT, Blockchain and IPFS for data management security. *Egypt. Inform. J.* **2022**, *23*, 329–343. [CrossRef]
4. Hossan, M.S.; Khatun, M.L.; Rahman, S.; Reno, S.; Ahmed, M. Securing ride-sharing service using IPFS and hyperledger based on private blockchain. In Proceedings of the 2021 24th International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 18–20 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
5. Lin, Y.; Zhang, C. A method for protecting private data in IPFS. In Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Dalian, China, 5–7 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 404–409.
6. Battah, A.A.; Madine, M.M.; Alzaabi, H.; Yaqoob, I.; Salah, K.; Jayaraman, R. Blockchain-based multi-party authorization for accessing IPFS encrypted data. *IEEE Access* **2020**, *8*, 196813–196825. [CrossRef]
7. Huang, H.S.; Chang, T.S.; Wu, J.Y. A secure file sharing system based on IPFS and blockchain. In Proceedings of the 2nd International Electronics Communication Conference, Singapore, 8–10 July 2020; pp. 96–100.
8. Sun, J.; Yao, X.; Wang, S.; Wu, Y. Blockchain-based secure storage and access scheme for electronic medical records in IPFS. *IEEE Access* **2020**, *8*, 59389–59401. [CrossRef]
9. Kang, P.; Yang, W.; Zheng, J. Blockchain private file storage-sharing method based on IPFS. *Sensors* **2022**, *22*, 5100. [CrossRef] [PubMed]
10. Uddin, M.N.; Hasnat, A.H.M.A.; Nasrin, S.; Alam, M.S.; Yousuf, M.A. Secure file sharing system using blockchain, ipfs and pki technologies. In Proceedings of the 2021 5th International Conference on Electrical Information and Communication Technology (EICT), Khulna, Bangladesh, 17–19 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5.
11. Li, W.; Guo, H.; Nejad, M.; Shen, C.C. Privacy-preserving traffic management: A blockchain and zero-knowledge proof inspired approach. *IEEE Access* **2020**, *8*, 181733–181743. [CrossRef]
12. Bellés-Muñoz, M.; Isabel, M.; Muñoz-Tapia, J.L.; Rubio, A.; Baylina, J. Circom: A circuit description language for building zero-knowledge applications. *IEEE Trans. Dependable Secur. Comput.* **2022**, 20, 4733–4751. [CrossRef]
13. Circom Official Website. Available online: https://iden3.io/circom (accessed on 24 March 2024).
14. Snarkjs Github Repository. Available online: https://github.com/iden3/snarkjs (accessed on 4 June 2024).
15. ZCash. Available online: https://z.cash/ (accessed on 12 July 2024).
16. Ma, W.; Xiong, Q.; Shi, X.; Ma, X.; Jin, H.; Kuang, H.; Gao, M.; Zhang, Y.; Shen, H.; Hu, W. Gzkp: A gpu accelerated zero-knowledge proof system. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, Vancouver BC Canada, 25–29 March 2023; pp. 340–353.
17. Han, C.; Shibano, K.; Ozaki, W.; Osaki, K.; Haraguchi, T.; Hirahara, D.; Kimura, S.; Kobayashi, Y.; Mogi, G. All-in-one platform for AI R&D in medical imaging, encompassing data collection, selection, annotation, and pre-processing. *In Proceedings of the Medical Imaging 2024: Imaging Informatics for Healthcare, Research, and Applications, San Diego, CA, USA, 18–23 February 2024*; SPIE: Bellingham, WA, USA, 2024; Volume 12931, pp. 311–315.
18. Miano, J. *Compressed Image File Formats: Jpeg, png, gif, xbm, bmp*; Addison-Wesley Professional: Boston, MA, USA, 1999.

19. Content Identifiers (CIDs). Available online: https://docs.ipfs.tech/concepts/content-addressing/#cids-are-not-file-hashes (accessed on 24 March 2024).
20. circom-chacha20 Github Repository. Available online: https://github.com/reclaimprotocol/circom-chacha20 (accessed on 24 March 2024).
21. The 8 × 8 dot Japanese Font "Misaki Font". Available online: https://littlelimit.net/misaki.htm (accessed on 11 June 2024). (In Japanese)
22. Grassi, L.; Khovratovich, D.; Rechberger, C.; Roy, A.; Schofnegger, M. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Vancouver, BC, Canada, 11–13 August 2021; pp. 519–535.
23. Circomlib Github Repository. Available online: https://github.com/iden3/circomlib (accessed on 21 March 2024).
24. Circomlibjs Github Repository. Available online: https://github.com/iden3/circomlibjs (accessed on 21 March 2024).
25. Khovratovich, D. Encryption with Poseidon. 2019. Available online: https://drive.google.com/file/d/1EVrP3DzoGbmzkRmYnyEDcIQcXVU7GlOd/view (accessed on 19 July 2024).
26. Poseidon-Encryption-Circom2 Github Repository. Available online: https://github.com/Shigoto-dev19/poseidon-encryption-circom2 (accessed on 21 March 2024).
27. Zhang, Y.; Wang, S.; Zhang, X.; Dong, J.; Mao, X.; Long, F.; Wang, C.; Zhou, D.; Gao, M.; Sun, G. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 14–18 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 416–428.
28. Kothapalli, A.; Setty, S.; Tzialla, I. Nova: Recursive zero-knowledge arguments from folding schemes. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 15–18 August 2022; Springer: Cham, Switzerland, 2022; pp. 359–388.