Self-sovereign and Secure Data Sharing Through Docker Containers for Machine Learning on Remote Node

Jungchul Seo, Younggyo Lee and Young Yoon*

Department of Computer Engineering, Hongik University, Seoul South Korea
E-mail: jcseo00@mail.hongik.ac.kr; young63571873@gmail.com; young.yoon@hongik.ac.kr
* *Corresponding Author*

Received 15 March 2024; Accepted 19 June 2024

Abstract

Collecting personal data from various sources and using it for machine learning (ML) is prevalent. However, there are increasing concerns about the monopolization and potential breach of private data by greedy and malicious organizations. Interest in Web 3.0 systems is on the rise as an alternative. These systems aim to guarantee the self-sovereignty of personal data in a decentralized setting. Users can share data with others directly for fair compensation. Nevertheless, malicious remote users can still violate the integrity and confidentiality of personal data. Therefore, this paper proposes a novel method of preventing unwanted leakage and counterfeiting of the private data lent on the premise of remote users. This paper focuses on the decentralized nature of Web 3.0 to leverage existing personal storage so that the burden of collecting secure data is relieved. Data owners create a lightweight Docker container to encapsulate their private data sources. The data owners generate another container to be deployed on a remote premise

Journal of Web Engineering, Vol. 23_5, 637–656.
doi: 10.13052/jwe1540-9589.2352

for taking and executing any ML algorithms remote users create. Between the containers forming a distributed trusted execution environment (TEE), data are read through a secure channel. Since the TEE is strictly controlled by the data owner, no malicious ML application can leak or breach the private information. This paper explains the engineering details of how this new method is realized.

Keywords: Self-sovereignty, trusted execution environment, data sharing, containers, Web3.0.

1 Introduction

Collecting personal data from various sources and using it for machine learning (ML) purposes is prevalent. However, there are increasing concerns about the monopolization and potential breach of private data by greedy and malicious organizations. As an alternative, interest in Web 3.0 systems is increasing. Web 3.0 aims to offer more diversified machine learning approaches with the recent advancements of distributed environment control algorithms and hardware technologies for creating various personalized services [1].

Data owners can lend their data directly to others remotely for fair compensation. However, ensuring the self-sovereignty of data is challenging [2,3]. Data owners are not free from concerns about unauthorized access, breach of private information, unwanted leakage, and counterfeit by remote users.

Data owners can consider several techniques for protecting their data through de-identification, differential privacy, federated learning, and homomorphic encryption [4–7] when sharing their data with remote users. However, these techniques can cause loss of information and lead to reduced utilization and lower accuracy of the analysis data, limiting its usefulness eventually [8,9]. In particular, unstructured data is challenging to preprocess and to extract features due to the lack of a clear structure.

Data collection preprocessed with the security measures above can incur significant communication, storage, and operation costs. Consistent quality control and scheduling of collected data among ML applications can be nontrivial, especially in a large-scale environment.

This paper focuses on the decentralized nature of Web 3.0 to leverage existing personal storage so that the burden of collecting secure data is relieved. Data owners create a lightweight Docker container to encapsulate their private data sources. The data owners generate another container to be deployed on a remote premise for taking and executing any ML algorithms remote users create. Between the containers forming a distributed trusted execution environment (TEE), data are read through a secure channel since the TEE is strictly controlled by the data owner; even a malicious ML application is blocked from leaking or breaching private information.

This methodology has two advantages. First, it can avoid complicated and costly data protection measures by enclosing original data sources and remote users' ML applications in a secure environment to preserve privacy and support highly accurate training through unprocessed data. Second, this approach can be scalable, as the data can be preprocessed using its distributed resources without accumulating in central storage.

The rest of the paper is structured as follows: Section 2 introduces related work; Section 3 presents background knowledge; Section 4 explains the design of our approach; Section 5 demonstrates a sample operation; finally, in Section 6, we conclude and discuss future studies.

2 Related Work

In Web 3.0, we envision personal data in various modes (e.g., voice, video, image, and text) that can be distributed over the network and shared across remote devices and servers for analytics and machine learning [1, 10]. However, there are concerns about securing the data and preserving privacy.

There are ongoing efforts to address such issues, including de-identification measures, homomorphic encryption, and distributed learning models. However, implementing these solutions in real-world settings is challenging due to the complexity of communication loads and additional implementation requirements.

2.1 De-dentification and Differential Privacy

Differential privacy is a mathematical anonymization technique that guarantees the difference between the result of processing personal information and the result of not using personal information below a certain level. To prevent abuse of personal information, noise insertion or deletion in processes like collecting, storing, processing, and sharing can maintain a certain level of change in query results due to data transformation, thereby controlling personal information exposure and quantifying the level of privacy protection [11].

Various theoretical studies have been conducted. However, the results have not yet been converted into practical solutions. The U.S. Census Bureau applied differential privacy to its 2020 census results. However, they argued that a large portion of the data could not be fundamentally disclosed and that limited data with privacy information obscured alone was not sufficient to draw meaningful conclusions [12].

2.2 Homomorphic Encryption

Homomorphic encryption allows data analysis without decryption so that encrypted data containing sensitive information can reliably be conveyed to various service environments with little concern about privacy breaches [13, 14]. Content-based publish/subscribe clients exchange messages through brokers by hiding sensitive information through a homomorphic re-encryption technique [15]. Smart contract [16] fulfillment can be verified with homomorphic encryption without revealing the contract details.

However, homomorphic encryption is currently limited regarding supported mathematical operations, making it difficult to perform large-scale complex analyses [17–19]. Despite the recent research efforts to improve accuracy and storage space efficiency, it falls short in supporting complex machine learning applications.

2.3 Distributed Learning Technology

Recently, various privacy-preserving distributed learning techniques have been studied, including federated learning [20, 21], which trains using distributed client-owned data and where a central server merges or aggregates the entire model, split learning [22], which learns by dividing neural networks into client and server parts, and combined split-fed learning [23, 24].

As these studies rely on transmitting and updating model parameters or data over the network, issues of communication load generation and increased bandwidth usage, security issues for model parameters, and the complexity of additional implementations for managing network communication and data transmission still need to be resolved. In addition, due to different computational and communication environments, it is unsuitable for real-time processing because of network topology and delay-induced asynchronous communication problems. Communication load costs increase when merging or aggregating learned models based on local updates to central servers. Scalability is limited in large environments, and the quality of data collected from local devices is inconsistent [25].

3 Background Knowledge

Our method utilizes Docker container technology to create a trusted execution environment that is logically independent and isolated from the host. We also implement authentication based on one-time password (OTP) technology to ensure confidentiality and integrity of shared data. Lastly, our method adopts HTTPS-based REST API technology for mutually safe and secure communication.

3.1 Docker-based Trusted Execution Environment

Existing trusted execution environment (TEE) [26, 27] technology provides physical isolation to ensure a higher level of data integrity and confidentiality than the rich execution environment(REE) that offers significantly more features and applications but is vulnerable to attacks [28].

Docker [29] is an open-source virtualization platform for container creation and management that abstracts the execution environment into containers, provides them as service units, and optimizes management with Kubernetes [30]. Docker does not include a separate operating system but relies on the kernel's function to isolate resources such as CPU, memory, block input/output, and network, allowing the operating system to have an independent process, file system, and network.

A container [31] is a type of software packaged as an image of the application and operating environment required for the software's execution environment. By creating and distributing new images without changing the execution environment, convenient management, easy expansion, and lightweight systems are guaranteed to run the same anytime, anywhere, and provide fundamental isolation.

In the proposed system, a Docker container can construct a logical TEE through resources isolated from the data user's host, provide confidentiality and integrity of shared data quickly and continuously in various environments, and operate and distribute independently. A Docker container eliminates the need for physical hardware to create a TEE. We chose not to rely on physical TEE, especially on the remote side, because it is not under the control of the data owner. Moreover, physical TEE can be limited in memory in practice.

3.2 REST API

The REST API, proposed by Roy Fielding and based on representational state transfer (REST), is a software protocol for efficiently managing service communication and interaction using HTTP methods such as create, read, update, and delete. In this model, HTTPS-based REST API communication authenticates users and ensures secure self-sovereignty for the data owners.

3.3 One-time Password

OTP [32] generates a unique password that can only be used once for security against authentication value leakage. OTP synchronization methods are mainly used as request–response, event synchronization, and time synchronization combinations. Synchronization based on time or events is the most prevalent approach. In the proposed model, OTP limits access to data. Different encryption and authentication security keys are assigned to individual users to ensure secure communication, confidentiality, and integrity of shared data.

4 System Design

This section presents a data-sharing system that provides owners with self-sovereignty of distributed data to ensure owners' rights and interests.
Our system has the following unique features:

- First, to prevent the abuse of data and the monopolization of collected data, a TEE is created to realize a secure space that is logically isolated and independent from the data user's host.
- Second, through Docker container technology configuration, installation efforts on the remote side for data deployment and analytics operation are minimized.
- Third, secure communication channels are being established, and data access control policies (ACPs) are being enforced to block access attempts by malicious users.
- Lastly, time-based OTP and HTTPS-based REST API technologies are being used to provide detailed user permissions.

4.1 Architecture and Interoperation Between Data Owners and Users

Our system comprises several modules, as illustrated in Figure 1. We describe the interaction based on the containers specified as follows (the symbol $\oplus$ denotes XOR operation):

- User Docker container and owner Docker container: UDC, ODC

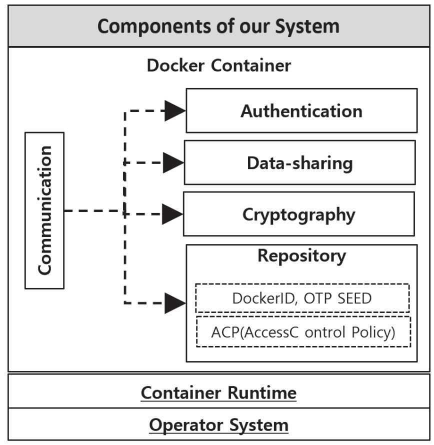


Figure 1 Components of our system.

- Data user identification information: $ID_{user's}$
- OTP seed by data user: $SEED_{user's}$
- Data access policy: ACP_{CRUD}(C: create, R: read, U: edit, D: delete)
- User OTP: $OTP_{user's} = H(SEED_{user's} \oplus TimeStamp)$
- Encryption keys: $SKey_{user's\ enc} = GEN\_KEY(VerifyOTP_{user's})$
- Decryption keys: $SKey_{user's\ dec} = GEN\_KEY(OTP_{user's})$
- Network file system: NFS

The communication module uses HTTPS-based REST API to establish a secure connection between data owners and data users. The authentication module uses $ID_{user's}$, $OTP_{User's}$, and ACP_{CRUD} to manage user authentication and access rights. The encryption module provides encryption and decryption algorithms that ensure the confidentiality and integrity of data by using different security keys for each user. The data-sharing module provides network-sharing capabilities. The storage module manages sensitive information such as $ID_{user's}$, $SEED_{user's}$, and ACP_{CRUD}.
Our system has two types of containers: User Docker container (UDC) and owner Docker container (ODC). These containers interact, as shown in Figure 2.

4.2 Implementation

The model consists of four stages: initialization for data sharing, user authentication, data sharing, and termination. Each stage operates within the UDC and ODC, a logically independent, trusted execution environment.

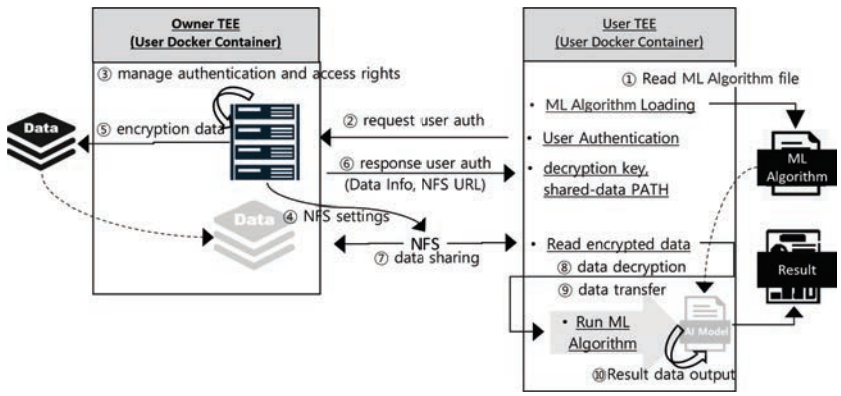


Figure 2 Interaction between data owner and data users.

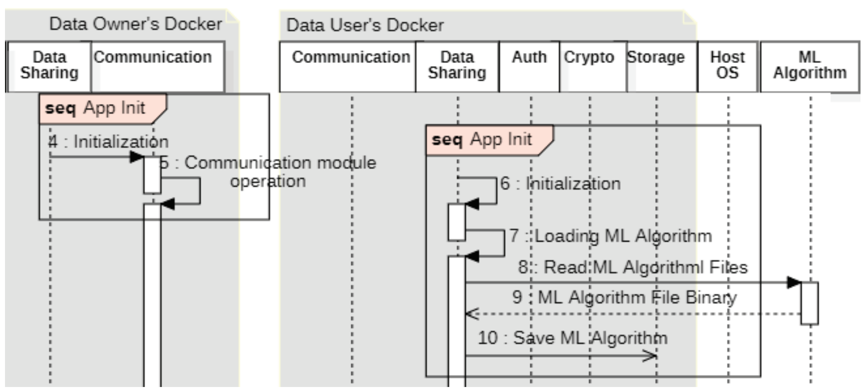


Figure 3 Initialization procedure

ODC first initiates HTTPS-based REST interface communication with the UDC. The UDC can employ AI-driven techniques to detect malware in the ML binaries and filter out malicious network packets to prevent data leakage [33]. It stores ML algorithms to leverage shared data as internal storage. Following the initialization, an ML is constructed with the data from ODC as training data. Figure 3 shows the detailed processing.

UDC and ODC perform user authentication as shown in Figures 4 and 5. The UDC calculates $OTP_{user's}$ using the pre-stored $ID_{user's}$, $SEED_{user's}$, and the current time and transmits authentication request information $(ID_{user's}, OTP_{user's})$ to the ODC for user authentication.

$$OTP_{user's} = H(SEED_{user's} \oplus TimeStamp). \tag{1}$$

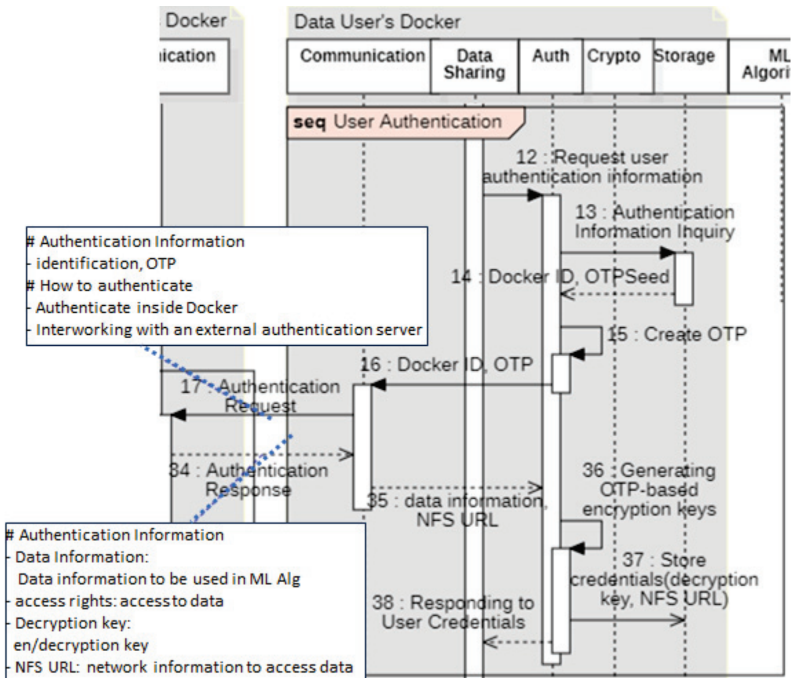


Figure 4 UDC's user authentication processing.

When the ODC receives a user authentication request from the UDC, it uses $ID_{user's}$ to query the repository for $SEED_{user's}$ and ACP_{CRUD}, uses $SEED_{User's}$ and the current time to calculate $VerifyOTP_{User's}$, and Authenticate users using the calculated $VerifyOTP_{User's}$ and the received $OTP_{User's}$ and control access to data based on the inquired ACP_{CRUD}.

$$VerifyOTP_{user's} = H(stored\ SEED_{user's} \oplus TimeStamp) \quad (2)$$

$$User's\ ACP_R = R : Read. \quad (3)$$

The ODC activates the NFS server to share data with authenticated users and creates an encryption key using the $VerifyOTP_{user's}$. It encrypts the data to be shared and delivers the shared data information(name, size, access rights) and NFS access information to the UDC.

$$SKey_{user's\ enc} = GEN\_Key(VerifyOTP_{user's}) \quad (4)$$

$$EncryptedData = E_{SKey_{user's\ enc}}(Data) \quad (5)$$

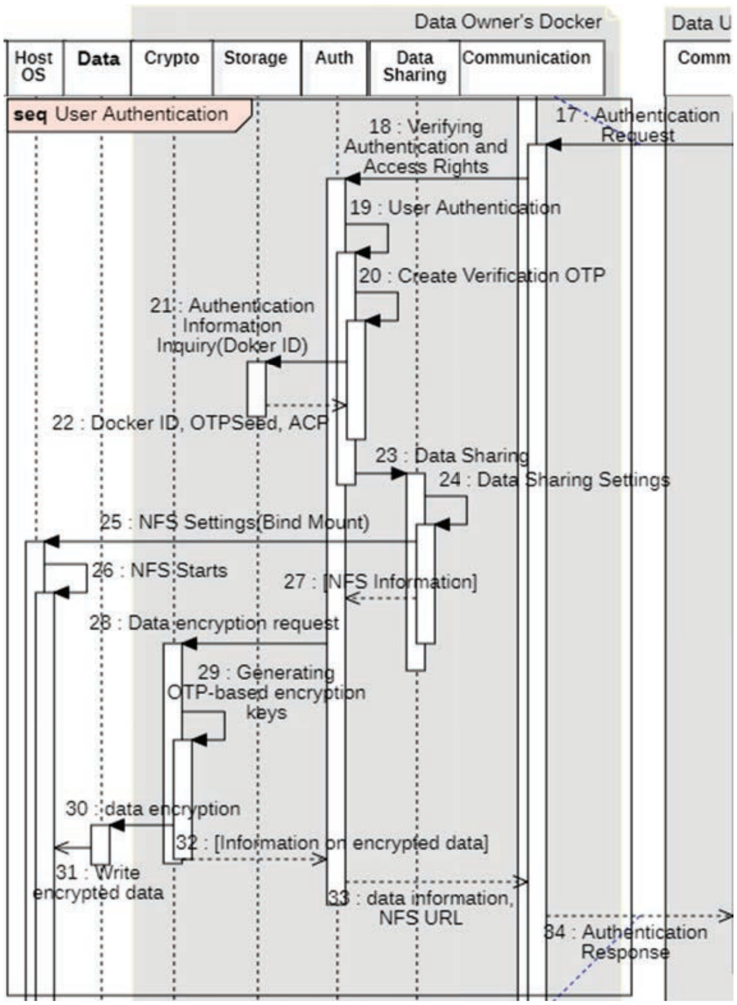


Figure 5 ODC's user authentication processing.

The UDC receives an authentication result from the ODC and calculates a decryption key.

$$SKey_{user's\ dec} = GEN\_Key(OTP_{user's}) \tag{6}$$

$$DecryptedData = D_{SKey_{user's}}(EncryptedData). \tag{7}$$

Following user authentication, the UDC connects to the ODC's NFS server to decrypt the encrypted shared data and execute the ML algorithm.

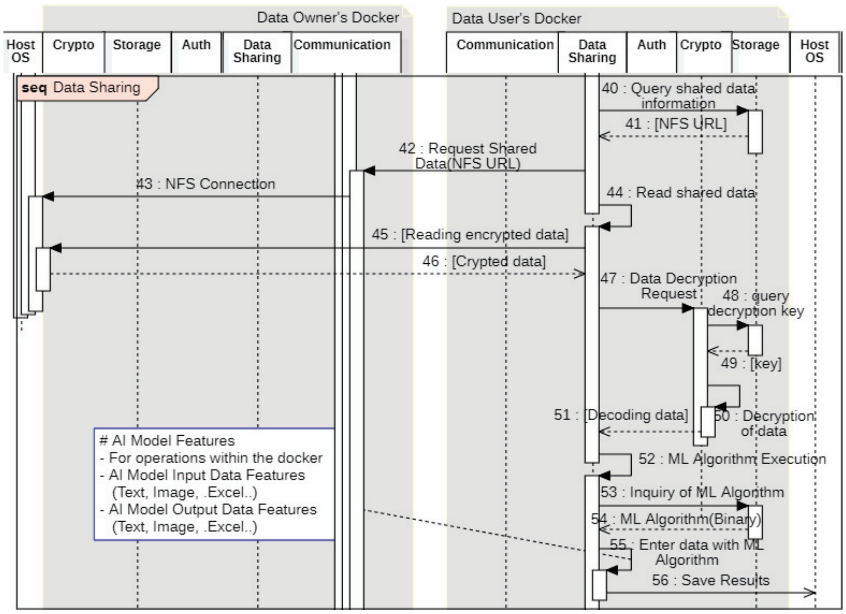


Figure 6 Data sharing processing.

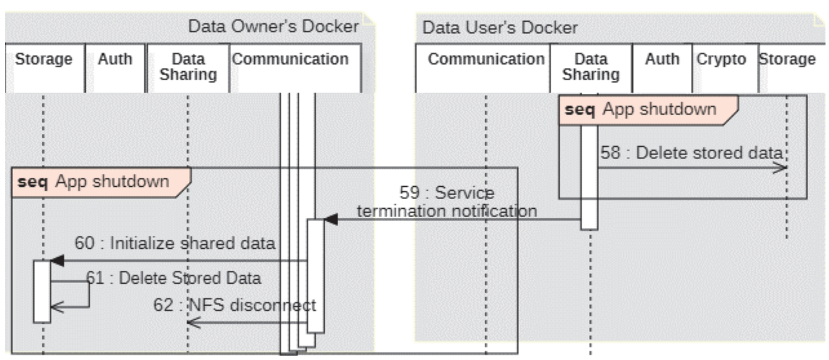


Figure 7 Termination procedure.

The result is checked inside the Docker and delivered safely to the host. Figure 6 shows the detailed processing.

When the UDC completes its operation or receives a data usage completion notification from the ODC, NFS information, decryption keys, and ML models are removed from the container. Figure 7 shows the detailed processing.

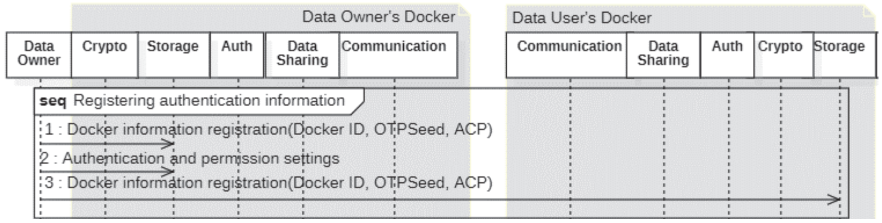


Figure 8 Authentication information registration.

Table 1 Setup of containers for testing

Docker container environment	
Operating system	Ubuntu 22.04
Programming language	Go 1.21.6, Echo(v4)
File system	NFS
Shared directory	/mnt/nfs_share

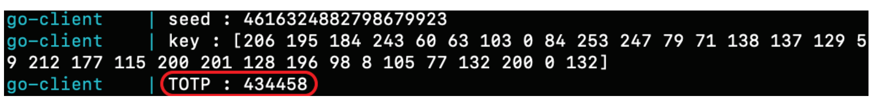


Figure 9 The UDC generates $OTP_{user's}$ and $SKey_{user's\ dec}$.

5 Demonstration

We demonstrate a sample operation between the data owner and a user. For this demonstration, we set $ID_{user's}$, $SEED_{user's}$, and ACP_{CRUD}, as shown in Figure 8.

The environment is shown in Table 1.

Figure 9 shows that UDC generates $OTP_{user's}$ unique for each user using $SEED_{user's}$ and the current time, and $SKey_{user's\ dec}$ for data decryption.

Figure 10 shows that the ODC generates $VerifyOTP_{user's}$ unique for each user using $SEED_{user's}$ and the current time, and $SKey_{user's\ enc}$ for data encryption.

Figure 11 shows the results of a typical NFS packet dump with data exposed and a packet dump of a proposed model with encrypted data.

Figure 12 shows that once the data usage is complete, the NFS link is disconnected, rendering the data inaccessible within the Docker container.

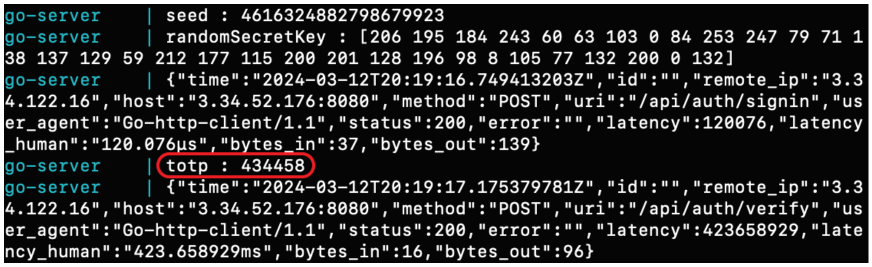


Figure 10 The ODC generates $VerifyOTP_{user's}$ and $SKey_{user's\ enc}$.

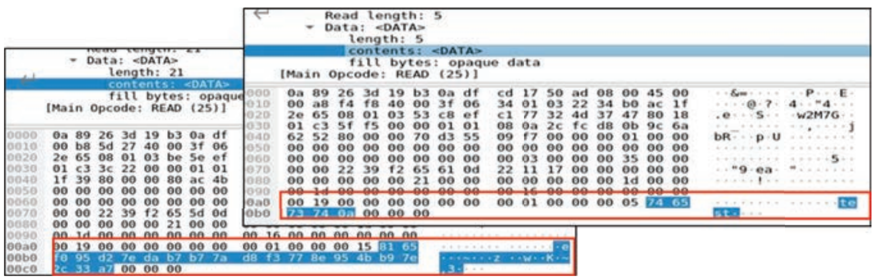


Figure 11 NFS packet dump.

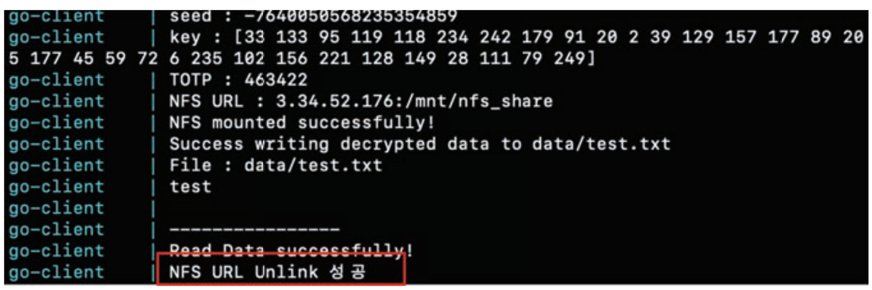


Figure 12 Disconnected NFS link.

Figure 13 demonstrates that when the proposed system is terminated, the NFS information, decryption key, and ML algorithm information are initialized, and the data is not stored.

Figure 14 shows that NFS mount procedures use the highest latency in UDC procedures.

Figure 15 shows that NFS mount procedures use the highest latency in ODC procedures.

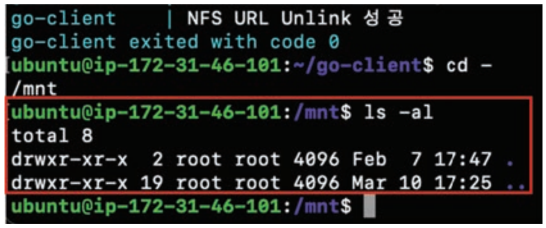


Figure 13 Initialized host's data.

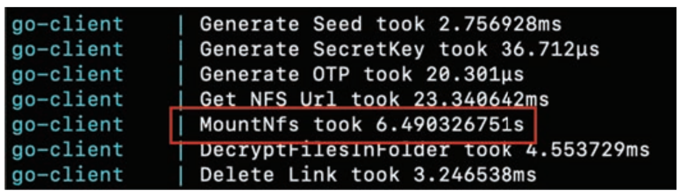


Figure 14 Latency by UDC's procedure.

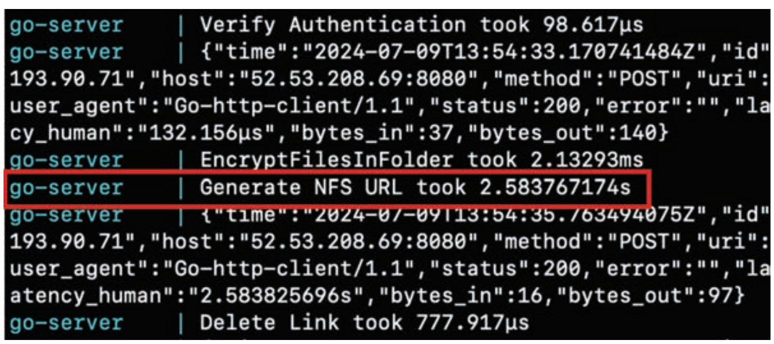


Figure 15 Latency by ODC's procedure.

6 Conclusion

This paper presented the self-sovereignty of data shared securely on the remote host within a logically isolated Docker container. Data stored in the NFS server on the owner-side Docker container (ODC) is encrypted on-demand with a time-based pseudo-random number as an OTP. The encrypted data is transferred via the REST interface to the user-side Docker container (UDC) for ML model training. Only the ML modeling outcome is returned to the UDC host, and the rest of the information, such as the OTP-based decryption keys, NFS information, and training data from ODC are removed. Upon completion of the data usage, ODC deactivates NFS. This methodology allows data owners to lend their data to remote users without concerns about privacy breaches and integrity violations.

In this paper, logical TEE for realizing a reliable execution environment with only software without hardware support is limited to Docker containers. In future work, we plan to apply various logical TEEs, such as KVM (kernel-based virtual machine), microkernels, and sandboxing, and minimize the latency of interaction procedures between data owners and data users. The right to access the data must be detailed to manage the owner's autonomy over the data in detail. Security and privacy infringement research is needed to minimize the threat of malicious ML algorithms that leak sensitive information out of the container-based logical trust environment.

Acknowledgement

This work was supported by 2024 Hongik University Innovation Support Program Fund and 2024 Hongik University Research Fund, by the MSIT (Ministry of Science and ICT), Korea under the ITRC (Information Technology Research Center) support program (RS-2023-00259099) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation, and by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00240211).

References

[1] Jungmin Kim and Kangho Bong. Survey on artificial intelligence industry. Technical report, IITP, 2023. https://spri.kr/posts/view/23578?code=sw_reports&s_year=&data_page=1 [Accessed: July 2, 2024].

[2] Magnus Redeker, Sören Volgmann, Florian Pethig, and Johannes Kalhoff. Towards data sovereignty of asset administration shells across value added chains. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1151–1154. IEEE, 2020.

[3] Atilla Aydın and Türksel Kaya Bensghir. Digital data sovereignty: towards a conceptual framework. In *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, pages 1–6. IEEE, 2019.

[4] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. Secure, privacy-preserving and federated machine

learning in medical imaging. *Nature Machine Intelligence*, 2(6):305–311, 2020.

[5] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[6] Sabrina Sicari, Alessandra Rizzardi, and Alberto Coen-Porisini. Insights into security and privacy towards fog computing evolution. *Computers & Security*, page 102822, 2022.

[7] Lizhi Sun, Shuocheng Wang, Hao Wu, Yuhang Gong, Fengyuan Xu, Yunxin Liu, Hao Han, and Sheng Zhong. Leap: Trustzone based developer-friendly tee for intelligent mobile apps. *IEEE Transactions on Mobile Computing*, 2022.

[8] Soo-Yong Shin. Issues and solutions of healthcare data de-identification: the case of south korea. *Journal of Korean Medical Science*, 33(5), 2018.

[9] Emily M Weitzenboeck, Pierre Lison, Malgorzata Cyndecka, and Malcolm Langford. The gdpr and unstructured data: is anonymization possible? *International Data Privacy Law*, 12(3):184–206, 2022.

[10] Young Yoon, Dae-hyun Ban, Sung-Won Han, Hong-Uk Woo, Eun-ho Heo, Sang-Ho Shin, Jung-kyuen Lee, and Dong-hyeok An. Terminal, cloud apparatus, driving method of terminal, method for processing cooperative data, computer readable recording medium, January 18 2022. US Patent 11,228,653.

[11] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vand. J. Ent. & Tech. L.*, 21:209, 2018.

[12] Steven Ruggles, Catherine Fitch, Diana Magnuson, and Jonathan Schroeder. Differential privacy and census data: Implications for social and economic research. In *AEA papers and proceedings*, volume 109, pages 403–408. American Economic Association 2014 Broadway, Suite 305, Nashville, TN 37203, 2019.

[13] Craig Gentry. Computing arbitrary functions of encrypted data. *Communications of the ACM*, 53(3):97–105, 2010.

[14] Kundan Munjal and Rekha Bhatia. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex & Intelligent Systems*, 9(4):3759–3786, 2023.

[15] Young Yoon and Jaehoon Kim. Homomorphic matching on publish/subscribe brokers based on simple integer partition and factorization for secret forwarding. In *Proceedings of the 23rd International Middleware Conference Demos and Posters*, pages 11–12, 2022.

[16] Young Yoon and Juno Moon. Verifying the integrity of private transaction information in smart contract using homomorphic encryption. In *2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 38–40. IEEE, 2019.

[17] Wonkyung Jung, Eojin Lee, Sangpyo Kim, Jongmin Kim, Namhoon Kim, Keewoo Lee, Chohong Min, Jung Hee Cheon, and Jung Ho Ahn. Accelerating fully homomorphic encryption through architecture-centric analysis and optimization. *IEEE Access*, 9:98772–98789, 2021.

[18] Youngjin Bae, Jung Hee Cheon, Wonhee Cho, Jaehyung Kim, and Taekyung Kim. Meta-bts: Bootstrapping precision beyond the limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 223–234, 2022.

[19] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. Hermes: Efficient ring packing using mlwe ciphertexts and application to transciphering. In *Annual International Cryptology Conference*, pages 37–69. Springer, 2023.

[20] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[21] Federated Learning. Collaborative machine learning without centralized training data. *Publication date: Thursday, April*, 6, 2017.

[22] Otkrist Gupta and Ramesh Raskar. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8, 2018.

[23] Zongshun Zhang, Andrea Pinto, Valeria Turina, Flavio Esposito, and Ibrahim Matta. Privacy and efficiency of communications in federated split learning. *IEEE Transactions on Big Data*, 2023.

[24] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493, 2022.

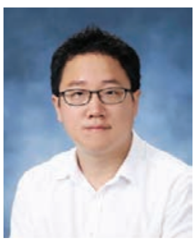
[25] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and

open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.
[26] globalplatform.org. Globalplatform specifications archive. https://globalplatform.org/specs-library/?filter-committee=tee [Accessed: 02.22.24].
[27] trustedfirmware.org. Op-tee documentation. https://optee.readthedocs.io [Accessed: 02.22.24].
[28] Nezer Jacob Zaidenberg, Raz Ben Yehuda, and Roee Shimon Leon. Arm hypervisor and trustzone alternatives. *Encyclopedia of Criminal Activities and the Deep Web*, pages 1150–1162, 2020.
[29] Wikipidia. Docker. https://en.wikipedia.org/wiki/Docker_(software) [Accessed: 02.22.24].
[30] Kubernetes.io. Kubernetes documentation. https://kubernetes.io/docs/home/ [Accessed: 02.22.24].
[31] docker.com. Use containers to build, share and run your applications. https://www.docker.com/resources/what-container [Accessed: 02.22.24].
[32] ietf. Totp: Time-based one-time password algorithm. https://datatracker.ietf.org/doc/html/rfc6238 [Accessed: 02.22.24].
[33] Hyeonmin Kim and Young Yoon. An ensemble of text convolutional neural networks and multi-head attention layers for classifying threats in network packets. *Electronics*, 12(20):4253, 2023.

Biographies

Jungchul Seo is a doctoral student at Hongik University. He also works as a developer at PHI Digital Healthcare. His research interests include computer security, artificial intelligence, distributed networks, and new Web 3.0 themes. Mr. Seo earned a master's degree in computer engineering from Pukyong National University in 2003.

Younggyo Lee is currently a senior undergraduate student at Hongik University. His research interests include cloud service security, network design, and new Web 3.0 problems. He joined the undergraduate program in computer engineering at Hongik University in 2019.



Young Yoon is an associate professor in computer engineering at Hongik University. He also serves as a CTO for Neouly Incorporated. His research interest is in distributed systems, middleware, cyber security, AI applications and emerging Web 3.0 issues. Yoon earned a B.A. and M.S. in computer sciences at the University of Texas at Austin in 2003 and 2006, respectively. He also earned his Ph.D. in computer engineering at the University of Toronto in 2013.