Gao, C.X., Wang, Y., & Huang, L. (2024). Privacy-preserving railway data sharing: A comparative study of homomorphic encryption schemes. In Li, E.Y. *et al.* (Eds.) *Proceedings of The International Conference on Electronic Business, Volume 24* (pp. 329-337). ICEB'24, Zhuhai, China, October 24-28, 2024

Privacy-Preserving Railway Data Sharing: A Comparative Study of Homomorphic Encryption Schemes

Cheng Xi Gao [1]
Ying Wang [2,*]
Lei Huang [3]

*Corresponding author
[1] School of Economics and Management, Beijing Jiaotong University, Beijing, China, gchengxi7@163.com
[2] School of Economics and Management, Beijing Jiaotong University, Beijing, China, ywang1@bjtu.edu.cn
[3] School of Economics and Management, Beijing Jiaotong University, Beijing, China, lhuang@bjtu.edu.cn

ABSTRACT

Railway data has been acknowledged as a vital asset for analyzing and enhancing the performance of railway enterprises and their supply chain partners. Therefore, data sharing among different organizational stakeholders has become a crucial issue. However, the proprietary nature of railway internal data poses challenges for cross-organizational data sharing, making a secure and privacy-preserving solution essential. In this context, the railway industry has paid attention to homomorphic encryption. The emerging technology provides a technical foundation for railway data sharing by enabling computations on encrypted data, ensuring confidentiality and trust in data sharing processes. This paper presents an application of homomorphic encryption to support privacy-preserving railway data sharing, conducting a comparative case study based on railway mileage calculation and evaluating the performance of three HE schemes (Paillier, BFV, CKKS). The results highlight the differences between the three schemes and offer guidance for their suitable application scenarios.

Keywords: Railway data, data sharing, homomorphic encryption, Paillier, BFV, CKKS.

INTRODUCTION

With the development of railway informatization, various business domains such as freight and passenger transport have accumulated extensive data resources. The surge in railway data has made it a crucial component in the development of the railway industry. These data not only help the railway enterprises in enhancing their internal transportation and operations, but also serves as a valuable resource for external entities such as government agencies, research institutions, and transportation companies, facilitating intermodal integration, safety improvement, customer experience enhancement, and more. Therefore, railway data holds significant asset value and is in high demand externally.

However, internal data within each railway company is typically considered proprietary and confidential, and its disclosure could potentially harm the entire organization's interests. The current method of railway data sharing is through techniques such as data interfaces and direct data files transfer, which may pose security risks like data leaks. Thus, privacy-preserving railway data sharing is required to ensure the secure exchange and extraction of value from railway data among trusted entities. Homomorphic encryption provides a technical foundation for privacy-preserving data sharing. This cryptographic technique allows computations to be performed on encrypted data without requiring decryption (Acar et al., 2019). This unique property makes homomorphic encryption particularly useful for privacy-preserving computation in scenarios where sensitive data needs to be processed securely (Naehrig et al., 2011; Moore et al., 2014; Li et al., 2020). In the context of railway data sharing, homomorphic encryption can be used to achieve confidentiality of data sharing, overcome barriers to railway data sharing, and enable trust between internal and external organizations.

Recent research on homomorphic encryption mainly focus on theoretical aspects, with limited attention given to practical applications (Yousuf et al, 2019). For scenarios with diverse computational requirements, practical solutions and systems addressing real-world problems are rare. In this paper, we seek to answer the following research question: *How do various homomorphic encryption schemes perform when applied to the railway data sharing scenario?* To this end, we conducted a case study on railway mileage calculation to illustrate the application in a real railway scenario, comparing the performance of three homomorphic encryption schemes.

The rest of this paper is structured as follows. The second section presents the related work about privacy-preserving data sharing and homomorphic encryption technology. The third section illustrates the application of homomorphic encryption to railway data sharing. In fourth section, we present a comparative evaluation of three homomorphic encryption (HE) schemes. Finally, the fifth section concludes this paper.

RELATED WORK

In this section, we provide an overview of current privacy-preserving railway data sharing and give some homomorphic encryption applications in other industries. Following this, we discuss the notable homomorphic encryption libraries that implement homomorphic encryption schemes. Finally, we delve into a detailed description of three specific HE schemes: Paillier, BFV, and CKKS, which will be utilized in the fourth section.

Privacy-Preserving Railway Data Sharing

With the increasing data sharing needs among railway companies and external companies, protecting sensitive information has become increasingly important. Currently, railway data sharing in China, particularly in intermodal transport (such as sea-rail transportation), is primarily facilitated through Electronic Data Interchange (EDI) platforms. These platforms enable the efficient exchange of key information such as the schedules of ship and train , freight rates, cargo status, and customs supervision. EDI is designed to enhance operational efficiency by ensuring the fast transmission of data between parties. However, its primary focus is on facilitating data exchange between parties to improve efficiency, rather than ensuring privacy protection. As a result, there has been relatively little research on privacy-preserving methods for railway data sharing. Homomorphic encryption, as one of the key technologies in privacy protection, has been widely used in various industries (Wood et al., 2020; Chen et al., 2021; Ali et al, 2023). By drawing on successful applications in other fields, the railway industry can enhance its data-sharing capabilities while preserving privacy.

In various industries, homomorphic encryption has been effectively used to protect sensitive data while enabling secure processing and decision-making. For instance, in healthcare, homomorphic encryption is employed to securely process sensitive patient data. Li et al. (2020) proposed a privacy-preserving medical diagnosis scheme, using the Paillier scheme and Euclidean distance to diagnose the disease of the patients while keeping patients' health data privacy. Li et al. (2022) proposed a secure decision-making scheme for IoMT applications, which used GSW scheme to handle linear functions effectively and securely. In finance, Balch et al. (2020) applied threshold fully homomorphic encryption to inventory matching, protecting the privacy of both buyers and sellers. Similarly, in the smart grid domain, Lin et al. (2022) proposed a power data sharing scheme based on blockchain and homomorphic encryption to improve data privacy.

For the railway industry, these applications have demonstrated the ability of homomorphic encryption to address privacy concerns while enabling secure and efficient data sharing.

Homomorphic Encryption Libraries

With the development of HE, more and more efficient and freely available HE libraries have emerged to simplify the application of theoretical concepts(Doan et al., 2023). Table 1 provides a summary of these implementations, detailing the libraries' name, supported schemes and programming language.

Table 1: Homomorphic encryption libraries.

Libraries	Supported Schemes	Language
SEAL	BFV, CKKS	C++
HElib	BGV, CKKS	C++
FHEW	FHEW	C++
HEAAN	CKKS	C++
PALISADE	BFV, BGV, CKKS, FHEW, TFHE	C++
Lattigo	BFV, CKKS	Go
Pyfhel	BFV, BGV, CKKS	Python
OpenFHE	BFV, BGV, CKKS, FHEW, TFHE, LMKCDEY	C++
TenSEAL	Python version of SEAL	Python
NuGet	C# version of SEAL	C#
node-seal	JavaScript version of SEAL	JavaScript/TypeScript

While there are numerous homomorphic encryption libraries available in various programming languages, such as C++ and C#, we chose to conduct our experiments using a Python library due to its compatibility with various data processing and analysis frameworks.

In this study, we selected the TenSEAL library for implementing homomorphic encryption due to its superior performance in ciphertext addition operations. Previous research has demonstrated that TenSEAL outperforms the Pyfhel library for both

CKKS and BFV schemes across all tested values of ciphertext dimension (Wiryen et al., 2024). The enhanced performance of TenSEAL in handling encrypted arithmetic operations makes it an ideal choice for our application, which requires efficient and secure processing of large-scale encrypted data.

Description Of Three HE Schemes

Homomorphic Encryption schemes primarily consist of four core algorithms: Key Generation (KenGen), Encryption, Decryption, and Evaluation (Eval). The subsequent sections will introduce and detail the algorithms of the Paillier, BFV, and CKKS schemes.

Paillier

In 1999, Paillier introduced a novel encryption scheme based on the composite residuosity problem (Rivest et al., 1978). It questions whether there exists an integer x such that $x^n \equiv a (\mathrm{mod}\ n^2)$ for a given integer a. Paillier scheme is Partially Homomorphic Encryption (PHE), which allows only one type of operation with an unlimited number of times. In other words, it allows only an unbounded number of additions but no multiplications. The following describes the main algorithm flow of Paillier scheme.

1. KeyGen Algorithm: For large primes p and q such that $gcd\,(pq, (p-1)(q-1)) = 1$, compute $n = pq$ and $\lambda = lcm(p-1, q-1)$. Then, select a random integer $g \in \mathbb{Z}^*_{n^2}$ by checking whether $gcd(n, L(g^{\lambda} \mathrm{mod} n^2)) = 1$, where the function L is defined as $L(u)=(u-1)/n$ for every u from the subgroup $\mathbb{Z}^*_{n^2}$ that is a multiplicative subgroup of integers modulo n^2. Finally, the public key is (n, g) and the secret key is a (p, q) pair.
2. Encryption Algorithm: Randomly generate r. Input plaintext m, and output the ciphertext $c = g^m r^n (\mathrm{mod} n^2)$.
3. Decryption Algorithm: For a proper ciphertext $c < n^2$, output the plaintext $m' = \frac{L(c^{\lambda} (\mathrm{mod} n^2))}{L(g^{\lambda} (\mathrm{mod} n^2))} \mathrm{mod} n$.
4. Homomorphic Addiction: Input ciphertext c and c', and compute and output the addition result $c_{add} = g^{c+c'}(r_1 \cdot r_1)^n (\mathrm{mod} n^2)$.

BFV

In 2012, Fan & Vercauteren (2012) made modifications to Brakerski's Fully Homomorphic Encryption scheme based on Learning With Errors (LWE) to work under the security assumption of RLWE, proposing the BFV scheme. BFV scheme is Fully Homomorphic Encryption (FHE), which allows an unlimited number of operations for an unlimited number of times. In other words, it allows both additions and multiplications. The BFV scheme involves multiple steps, as mentioned below.

1. KeyGen Algorithm: Choose appropriate security parameters, the modulus q and plaintext space R. Randomly generate $s \leftarrow \chi$, and set private key $sk \leftarrow (1, s)$. Randomly generate $a \leftarrow R_q, e \leftarrow \chi$, and set public key $pk \leftarrow (b, a)$, where $b = -(as + e) \cdot a$, a is from plaintext space, and e is from noise space.
2. Encryption Algorithm: Represent the plaintext m as an integer in the plaintext space R. Generate a small random error e from a distribution centered around 0. Output the ciphertext $c = (qm + 2e + s) \mathrm{mod} q$.
3. Decryption Algorithm: Output the plaintext $m' = (c \bmod q) \bmod R$.
4. Homomorphic Addiction: Input ciphertext c and c', and compute and output the addition result $c_{add} = c + c' \bmod q$.
5. Homomorphic Multiplication: Input ciphertext c and c', and compute and output the multiplication result $c_{mult} = c \cdot c' \mathrm{mod} q^2$.

CKKS

In 2017, Cheon et al. (2017) proposed a scheme of homomorphic encryption CKKS, which supports real number and complex number approximations. CKKS scheme is Fully Homomorphic Encryption (FHE). The following describes the main algorithm flow of CKKS.

Set safety parameters λ, and choose the power of two integers N. Set distributions $\chi_{key}, \chi_{err}, \chi_{enc}$ for key, learning with errors, and encryption on $R = \mathbb{Z}[X]/(X^N + 1)$ individually. To get a basic integer p and the number of levels L, set the modulus of the ciphertext $q_l = p^l (1 \le l \le L)$, where l is the level of ciphertext, then create an integer P at random, and output $pp = (N, \chi_{key}, \chi_{err}, \chi_{enc}, L, q_l)$:

1. KeyGen Algorithm: Randomly generate $s \leftarrow \chi_{key}$, and set private key $sk \leftarrow (1, s)$. Randomly generate $a \leftarrow U(R_{q_L})$ and $e \leftarrow \chi_{err}$, set $pk \leftarrow (-as + e, a) \in R^2_{q_L}$. Randomly generate $a' \leftarrow U(R_{q^2/L})$ and $e' \leftarrow \chi_{err}$, and set $evk \leftarrow (-a's + e' + q_L s^2, a') \in R_{q^2/L}$.
2. Encryption Algorithm: Randomly generate $r \leftarrow \chi_{enc}$ and $e_0, e_1 \leftarrow \chi_{err}$; input plaintext m, and output the ciphertext $c = r \cdot pk + (m + e_0, e_1) (\mathrm{mod} q_L)$.
3. Decryption Algorithm: For ciphertext of level l, compute and output the plaintext $m' =< c, sk > (mod q_l)'$.
4. Homomorphic Addiction: For the ciphertext c, c' of the same level l, input c and c', and compute and output the addition result $c_{add} = c + c' (\mathrm{mod} q_l)$.
5. Homomorphic Multiplication: $c = (c_0, c_1)$, $c' = (c'_0, c'_1) \in R^2_{q_l}$. Compute $(d_0, d_1, d_2) = (c_0 c'_0, c_0 c'_1 + c'_0 c_1, c_1 c'_1) (\mathrm{mod} q_l)$. Output the result of ciphertext multiplication $c_{mult} = (d_0, d_1) + \lfloor p^{-1} \cdot d_2 \cdot sk \rfloor (\mathrm{mod} q_l)$.

APPLYING HOMOMORPHIC ENCRYPTION TO RAILWAY DATA SHARING

In this section, we apply homomorphic encryption to railway data sharing, conducting a case study based on railway mileage calculation to illustrate the practically of our work.

Application Scenario

In the field of railway income liquidation, accurate prediction of mileage is crucial for settlement companies, as it directly impacts the computation of railway access charges. However, current mileage calculations present a set of challenges, including issues of trust and data security.

Railway mileage data, reported independently by each railway bureau, suffers from inconsistencies due to varying standards, methods, and human factors. This undermines the accuracy crucial for settlement processes, impacting fairness and reliability. Furthermore, data security poses a significant challenge, with sensitive statistics data often inaccessible to settlement companies, raising concerns about potential leaks.

Generalizing from railway mileage calculation, we consider four types of stakeholders in the scenario. Fig. 1 illustrates the method.

1. Railway Bureau: The railway bureau is the owner and sharer of railway data. It owns the railway data and hopes to share, utilize and realize data assets, but is worried that the reproducibility of data assets may lead to the secondary transfer of the shared data, and make yourself lose the dominance of data.
2. Settlement Company: The settlement company is the requestor and user of data, who lacks data and needs to obtain data from railway bureaus to liquid income. The settlement company submits data requests and computing services in the Railway Data Service Platform according to their needs, and need to obtain the results of joint computation in order to guide income liquidation.
3. Railway Data Service Platform: The Railway Data Service Platform is responsible for accepting requests from data demanders such as settlement companies and distributing the requests to data providers like the railway bureau. Once all of the involved data provider agrees to the data request, the Railway Data Service Platform will generate a homomorphic encryption key pair and send the public key to the data provider and the private key to the data demander. Since the Railway Data Service Platform only has the homomorphic encryption key pair and will not have access to both original data of data providers and ciphertext homomorphic result, it ensures the security of the data sharing process.
4. Cloud Server: The cloud server is responsible for receiving the ciphertext data uploaded by the data provider and performing homomorphic encryption calculations, and returns the generated ciphertext results to the data demander. Since the cloud server only has the ciphertext data and the ciphertext result, it ensures the security of the data sharing process.

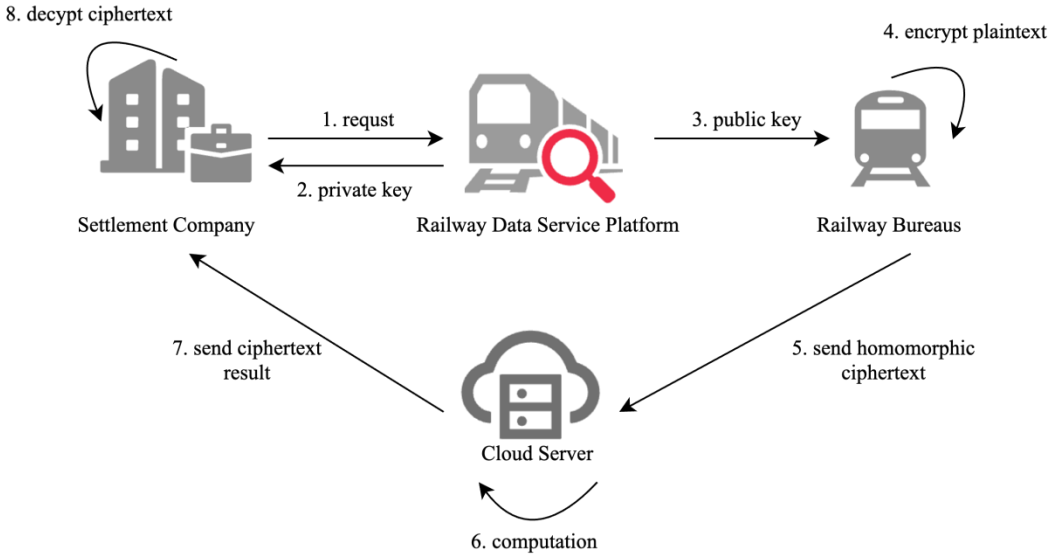

Figure 1: The multiple stakeholders in privacy-preserving railway data sharing.

Application Solution

Under the background of mileage calculation, a case study is conducted in to verify our proposed privacy-preserving data sharing solution. The whole data sharing process can be divided into four stages: application, initialization, encryption and computation, and decryption. Here, the homomorphic encryption algorithm takes the Paillier algorithm as an example. Fig. 2 illustrates each stage of this solution.

Application Stage:

First, the settlement company S need to submit a application request to obtain data usage rights. The application needs to include which railway bureaus' data is needed and what kind of data is required. When the application is sent to server, the server can automatically recognize which railway bureaus $R_i (i \in 1,2,3,\ldots n)$ are included in the application and sends the application to railway bureaus. Then, railway bureaus can review the application. Joint calculations can only be done when railway bureaus all agree.

Initialization Stage:

In order to achieve encryption, the Railway Data Service Platform will generate a homomorphic encryption public and private key pairs PK_{HE} and SK_{HE}. PK_{HE} will be sent to involved railway bureaus to encrypt data and SK_{HE} will be sent to settlement company to decrypt data.

Encryption and Computation Stage:

Railway bureaus R_i encrypt data $M_i = \{P_i, S_i\}$ homomorphially, where P_i represents the unique train number which is same in different railway bureaus and S_i represents the railway mileage data. $P_i = \begin{Bmatrix} p_1 \\ p_2 \\ \cdots \\ p_n \end{Bmatrix}$ and $S_i = \begin{Bmatrix} s_{11} & s_{12} & \cdots & s_{1k} \\ s_{21} & s_{22} & \cdots & s_{2k} \\ \cdots & & & \cdots \\ s_{n1} & s_{n2} & \ldots & s_{nk} \end{Bmatrix} (s_{pq} \in R, p = 1,2,\ldots n; q = 1,2,\ldots k)$. Encrypt each data s_{pq} sequentially using the homomorphic public key PK_{HE} to obtain ciphertext data $ctxt_{spq} = Enc(s_{pq}, PK_{HE})$($Enc$ is Homomorphic encryption algorithm). Encrypt data is $Ctxt_i = \begin{Bmatrix} ctxt_{11} & ctxt_{12} & \cdots & ctxt_{1k} \\ ctxt_{21} & ctxt_{22} & \cdots & ctxt_{2k} \\ \cdots & & & \cdots \\ ctxt_{n1} & ctxt_{n2} & \ldots & ctxt_{nk} \end{Bmatrix}$. So M_i becomes $C_i = \{P_i, Ctxt_i\}$.

Then, railway bureaus R_i send encrypted data C_i to Cloud Server. After receiving all encrypted data, the server can calculate railway mileage homomorphially and generate the result. The calculation process is below.

Perform additive homomorphic computation on the k fields data of each data record with the same train number in all ciphertexts. Assume that there are t railway bureaus with the same train number, taking the calculation of one field data in one train number as an example:

$$\begin{aligned} C_{result} &= ctxt_{s1} \cdot ctxt_{s2} \cdot \ldots \cdot ctxt_{st} \\ &= Enc_{HE}(ctxt_{s1}, PK_{HE}) \cdot Enc_{HE}(ctxt_{s2}, PK_{HE}) \cdots\cdots Enc_{HE}(ctxt_{st}, PK_{HE}) \\ &= g^{ctxt_{s1}+ctxt_{s2}+\ldots+ctxt_{st}} \cdot (r_1 \cdot r_2 \cdots\cdot r_t)^n (\mathrm{mod} n^2) \end{aligned} \quad (1)$$

After computing the all fields for all train numbers separately, we obtain the ciphertext data $Ctxt_{all}$.

Decryption Stage:

Once the Cloud Server calculates result successfully, it can send result of the ciphertext to settlement company. The settlement company can use their private key to decrypt the result. The settlement company S uses the private key SK_{HE} to decrypt each data in sequentially, taking decrypting $Ctxt$ as an example:

$$M_{result} = L(Ctxt^{\lambda} \mathrm{mod} n^2) \cdot \mu \mathrm{mod} n \quad (2)$$

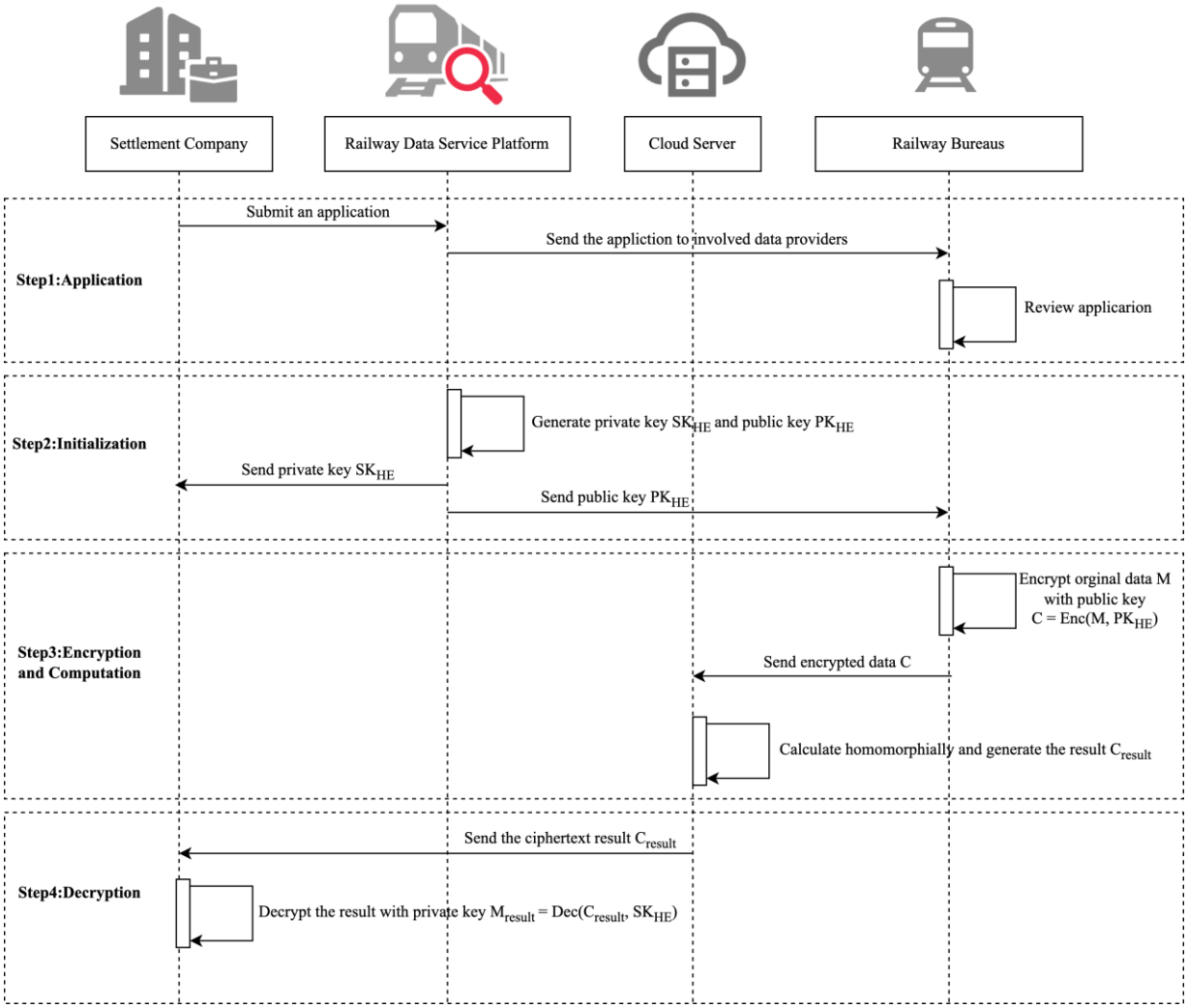

Figure 2: Sequence diagram of each stage of the method.

COMPARISONS BETWEEN THE IMPLEMENTATIONS OF VARIOUS SCHEMES

In this section, we will present a comparative study of three homomorphic encryption schemes: Paillier, BFV, and CKKS. To begin with the implementation of HE, we choose two Python libraries: phe and TenSEAL. We use the phe library to implement Paillier and use the TenSEAL library to implement BFV and CKKS. Experimental environment: the hardware used featured an Apple M2 with 8-core and 16 GB of memory. The software environment was MacOS, with Anaconda3, Python version 3.9, and the programming tool was Pycharm.

To implement the Paillier cryptosystem, the n_length parameter must be set in the phe library. For the BFV scheme in the TenSEAL library, two parameters need to be configured: poly_modulus_degree and plain_modulus. Implementing the CKKS scheme in the TenSEAL library requires setting four parameters: poly_modulus_degree, plain_modulus, coeff_mod_bit_sizes, and global_scale. There are five main HE parameters used by the schemes:

1. n_length (Paillier): The security of the Paillier scheme depends on the size of n. A larger n provides higher security but also increases computational overhead.
2. poly_modulus_degree (BFV, CKKS): This parameter determines the degree of the polynomial modulus. It directly affects the ciphertext size and the security level. A higher degree provides better security but increases computation and memory requirements.
3. plain_modulus (BFV): This is the modulus used for plaintext operations in the BFV schemes. It must be appropriately chosen to balance between the size of the plaintext space and the efficiency of the scheme. It affects batching capabilities for BFV.
4. coeff_mod_bit_sizes (CKKS): This parameter defines the bit sizes of the coefficient modulus. It controls the number of bits for each coefficient in the polynomial modulus, influencing the noise budget and overall security. Proper selection is crucial to ensure both security and the ability to perform multiple operations before re-encryption is necessary.
5. global_scale (CKKS): This is the scaling factor used to manage precision in CKKS. It determines how much the numbers are scaled during encryption. A larger scale provides higher precision but also increases the ciphertext size and computational complexity.
6.

We compare the performance of three different homomorphic encryption algorithms Paillier, BFV, and CKKS, from ciphertext size, encryption time, decryption time, and addition computation time. To ensure the reliability of our results, each experiment was conducted 50 times. However, due to the fundamental differences in three schemes, a direct one-to-one comparison of n_length and poly_modulus_degree is not straightforward. Instead, we perform experiments on Paillier with n_length alone and compare BGV and CKKS with poly_modulus_degree.

Table 2 presents the performance of the Paillier scheme for different key sizes. As key size increases, the time required for key generation, encryption, and decryption increases significantly. For example, key generation time increases from 17.6 ms (256 bits) to 1854.1 ms (2048 bits). So for applications requiring frequent encryption and decryption, smaller key sizes might be more practical if security requirements permit. The time required for homomorphic addition remains relatively low even for larger key sizes, suggesting that Paillier is efficient for operations that require frequent additions. Larger key sizes result in significantly larger ciphertexts, which can impact storage and communication efficiency.

Table 2: Paillier addiction performance.

n_length	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)	Addition (ms)	Ciphertext Size (bit)
256	17.6	66.8	31.6	0.3	15382.6
512	38.9	271.3	99.3	0.8	30785.8
1024	264.8	1796.8	616.3	1.9	61621.1
2048	1854.1	12160.2	3438.1	5.9	123274.2

Fig 3 shows the comparison between BFV and CKKS. When the polynomial modulus degree is the same, compare the encryption time, decryption time, key generation time, and ciphertext size of different schemes. Encryption time compared between the two schemes which shows that when the size of vectors was 4096, 8192 the two schemes have similar time consumption. When the vector size increases, CKKS shows better scalability in encryption time compared to BFV, making it more efficient for larger data sizes. Likewise, CKKS outperforms BFV in decryption time and key generation time, particularly as the vector size increases. Besides, CKKS produces significantly smaller ciphertexts compared to BFV, which means CKKS reduces storage and transmission overhead.

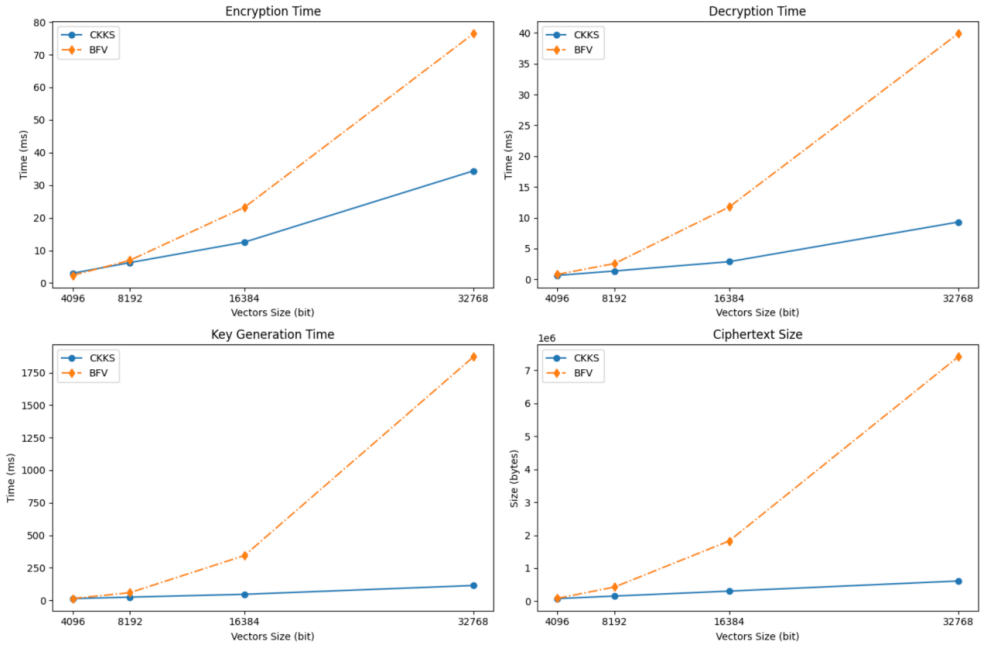

Figure 3: Comparison of BFV and CKKS.

Table 3 outlines the performance for addition and multiplication HE operations in both BFV and CKKS schemes at various poly_modulus_degree settings. CKKS generally exhibits faster addition times compared to BFV, making it more suitable for applications with frequent addition operations. Besides, CKKS is more efficient for multiplication operations, particularly at lower vector size settings. BFV's multiplication time increases more sharply with higher degrees, indicating higher computational overhead for such operation.

Table 3: Comparison of BFV and CKKS addition and multiplication operations.

HE Parameters		BFV (ms)		CKKS (ms)	
		Addition	Multiplication	Addition	Multiplication
poly_modulus_degree	4096	0.1547	11.0837	0.1033	2.2755
	8192	0.3324	25.8498	0.1425	5.7745
	16384	4.8757	126.8859	0.2622	6.3834
	32768	6.4437	572.1207	0.7885	32.9831
plain_modulus	65537	65537		/	
coeff_mod_bit_sizes	[40,20,40]	/		[40,20,40]	
global_scale	2^20	/		2^20	

To sum up, we compare the performance of the three homomorphic encryption schemes—Paillier, BFV, and CKKS—across several dimensions: ciphertext size, encryption time, decryption time, and addition operation time. While Paillier's simplicity makes it efficient for addition operations, its performance significantly degrades with increased key sizes due to higher computational overhead. On the other hand, CKKS demonstrates better scalability and outperforms BFV in both encryption and decryption time, particularly with larger datasets. BFV performs well for smaller datasets but suffers from increased computational overhead as the data size increases, particularly in multiplication operations. CKKS, with its efficient handling of both addition and multiplication operations and smaller ciphertext sizes, is more suited for applications that require frequent operations on large datasets. The smaller ciphertext size generated by CKKS further enhances its efficiency for storage and communication, making it a practical choice for applications handling large-scale data. In practical railway income liquidation scenarios, CKKS's scalability makes it a favorable choice.

One of the key limitations observed is the significant increase in computational overhead as key sizes grow in the Paillier and BFV schemes. While CKKS demonstrates better scalability, it also incurs higher computational costs for larger ciphertexts. Additionally, key size directly impacts both security and performance, making it critical to balance security requirements with computational feasibility in real-world deployments.

In real-world deployment, challenges such as model calibration and parameter sensitivity must be addressed. For example, choosing the appropriate poly_modulus_degree is critical for both BFV and CKKS schemes, as it affects both the security and performance of the system. Integration with existing railway systems poses additional complexities, particularly in maintaining efficient performance while ensuring data privacy at scale.

CONCLUSION

This paper presents a privacy-preserving method for railway data sharing based on homomorphic encryption. By leveraging homomorphic encryption, the method ensures that data demanders can obtain encrypted results without accessing the original data, thereby addressing the data protection needs of the data providers. In addition, this paper presents a comparative evaluation of three homomorphic encryption schemes, providing insights into their suitability for railway mileage calculation scenario. The applicability of our method is demonstrated through a practical application scenario, illustrating its potential for real-world use.

However, there are certain limitations to this approach. While homomorphic encryption offers a robust solution for privacy preservation, its computational overhead remains a challenge, particularly in large-scale railway data exchanges. Additionally, the security of the system depends on the correct implementation of encryption protocols, which may require further investigation to ensure scalability and performance in broader contexts.

Future work will focus on identifying more diverse railway data-sharing scenarios to further validate and enhance the proposed method. Furthermore, we plan to conduct more extensive evaluations in different railway environments to ensure the method's adaptability and effectiveness in various practical settings. This will also include an exploration of potential integration with other privacy-preserving technologies to enhance the robustness of our approach.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China [grant number 52172311, U2268202]

REFERENCES

Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur), 51*(4), 1-35. https://doi.org/10.1145/3214303

Ali, A., Al-Rimy, B. A. S., Alsubaei, F. S., Almazroi, A. A., & Almazroi, A. A. (2023). HealthLock: Blockchain-based privacy preservation using homomorphic encryption in internet of things healthcare applications. *Sensors*, *23*(15), 6762. https://doi.org/10.3390/s23156762

Balch, T., Diamond, B. E., & Polychroniadou, A. (2020). SecretMatch: Inventory matching from fully homomorphic encryption. In *Proceedings of the First ACM International Conference on AI in Finance* (pp. 1-7). https://doi.org/10.1145/3383455.3422569

Chen, C., Zhou, J., Wang, L., Wu, X., Fang, W., Tan, J., ... & Hong, C. (2021). When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (pp. 2652-2662). https://doi.org/10.1145/3447548.3467210

Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Part I 23* (pp. 409-437). https://doi.org/10.1007/978-3-319-70694-8_15

Doan, T. V. T., Messai, M. L., Gavin, G., & Darmont, J. (2023). A survey on implementations of homomorphic encryption schemes. *The Journal of Supercomputing*, *79*(13), 15098-15139.https://doi.org/10.1007/s11227-023-05233-z

Ducas, L., & Micciancio, D. (2015). FHEW: Bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques* (pp. 617-640). https://doi.org/10.1007/978-3-662-46800-5_24

Fan, J., & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*. Retrieved from https://eprint.iacr.org/2012/144 (accessed 8 November 2024).

Li, C., Yang, L., Yu, S., Qin, W., & Ma, J. (2022). SEMMI: Multi-party security decision-making scheme for linear functions in the internet of medical things. *Information Sciences, 612*, 151-167. https://doi.org/10.1016/j.ins.2022.08.102

Li, D., Liao, X., Xiang, T., Wu, J., & Le, J. (2020). Privacy-preserving self-serviced medical diagnosis scheme based on secure multi-party computation. *Computers & Security, 90*, 101701. https://doi.org/10.1016/j.cose.2019.101701

Li, J., Kuang, X., Lin, S., Ma, X., & Tang, Y. (2020). Privacy preservation for machine learning training and classification based on homomorphic encryption schemes. *Information Sciences, 526*, 166-179. https://doi.org/10.1016/j.ins.2020.03.041

Lin, Y., Liu, G., Wen, J., Hua, K., Jin, F., Hu, Y., ... & Zhang, Q. (2022). Power data blockchain sharing scheme based on homomorphic encryption. In *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (Vol. 5, pp. 625-629). https://doi.org/10.1109/IMCEC55388.2022.10020058

Moore, C., O'Neill, M., O'Sullivan, E., Doröz, Y., & Sunar, B. (2014). Practical homomorphic encryption: A survey. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 2792-2795). https://doi.org/10.1109/ISCAS.2014.6865753

Naehrig, M., Lauter, K., & Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? *In Proceedings of the 3rd ACM workshop on Cloud computing security workshop* (pp. 113-124). https://doi.org/10.1145/2046660.2046682

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*(2), 120-126. https://doi.org/10.1145/359340.359342

Wiryen, Y. B., Vigny, N. W. A., Joseph, M. N., & Aimé, F. L. (2024). A Comparative Study of BFV and CKKs Schemes to Secure IoT Data Using TenSeal and Pyfhel Homomorphic Encryption Libraries. *International Journal of Smart Security Technologies (IJSST), 10*(1), 1-17. https://doi.org/10.4018/IJSST.333852

Wood, A., Najarian, K., & Kahrobaei, D. (2020). Homomorphic encryption for machine learning in medicine and bioinformatics. *ACM Computing Surveys (CSUR), 53*(4), 1-35. https://doi.org/10.1145/3394658

Yousuf, H., Lahzi, M., Salloum, S. A., & Shaalan, K. (2020). Systematic review on fully homomorphic encryption scheme and its application. *Recent Advances in Intelligent Systems and Smart Applications*, 537-551. https://doi.org/10.1007/978-3-030-47411-9_29