A Privacy-Preserving Face Recognition Scheme Combining Homomorphic Encryption and Parallel Computing

Gong Wang[1], Xianghan Zheng[1], Lingjing Zeng[2], and Weipeng Xie[1(✉)]

[1] Fuzhou University, Fuzhou, China
2431455652@qq.com
[2] Fujian Chuanzheng Communicatians College, Fuzhou, China

Abstract. Face recognition technology is widely used in various fields, such as law enforcement, payment systems, transportation, and access control. Traditional face authentication systems typically establish a facial feature template database for identity verification. However, this approach poses various security risks, such as the risk of plaintext feature data stored in cloud databases being leaked or stolen. To address these issues, in recent years, a face recognition technology based on homomorphic encryption has gained attention. Based on homomorphic encryption, face recognition can encrypt facial feature values and achieve feature matching without exposing the feature information. However, due to the encryption, face recognition in the ciphertext domain often requires considerable time. In this paper, we introduce the big data stream processing engine Flink to achieve parallel computation of face recognition in the ciphertext domain based on homomorphic encryption. We analyze the security, accuracy, and acceleration of this approach. Ultimately, we verify that this approach achieves recognition accuracy close to plaintext and significant efficiency improvement.

Keywords: Homomorphic Encryption · Face recognition · Flink · Privacy Protection · Data Flow

1 Introduction

As the era of the Internet of Things approaches, various smart sensing devices are widely used, and face recognition systems are widely applied in areas such as law enforcement, payment, transportation, access control, and more. Compared to traditional identity authentication mechanisms, face recognition features the advantages of being less prone to forgetting or damage and offers convenient usage, effectively enhancing the efficiency of authentication. One common attack on face recognition is the use of facial feature template libraries. For instance, attackers can reconstruct the original face image to successfully bypass the authentication or infer personal attributes like age and gender.

Since data involves sensitive information, uploading data to the cloud poses risks of leakage. An effective solution is to encrypt the data before uploading it,

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024
J. Vaidya et al. (Eds.): AIS&P 2023, LNCS 14510, pp. 38–52, 2024.
https://doi.org/10.1007/978-981-99-9788-6_4

which can prevent data leaks, but it hinders feature matching in the ciphertext domain. General encryption algorithms can only perform encryption, and it has become a hot topic in academia to find a solution for performing computations on encrypted data.

Enabling computation on encrypted data has been a challenging issue that has perplexed many scholars until the advent of Fully Homomorphic Encryption (FHE) schemes. FHE can perfectly solve this problem as it allows computations on encrypted data. Subsequently, many scholars, inspired by his work, have gradually developed various FHE schemes, enriching the field of encrypted computation and providing theoretical support for achieving face recognition in the ciphertext domain.

In comparison to the existing homomorphic encryption face recognition algorithms, which are extremely time-consuming, this paper focuses on the challenges faced by face recognition in the ciphertext space, such as low efficiency and massive computational overhead. To address these challenges, the paper proposes a data stream computing framework for face privacy protection. It encrypts face data using the CKKS homomorphic encryption scheme and processes the data as a data stream using Flink, a stream processing framework. Flink features real-time processing, multi-threading, and stateful data processing, dividing the stream data processing into two Map-Reduce stages. The proposed framework aims to protect the facial ciphertext information stored in the database, enabling approximate homomorphic encryption for matching calculations while utilizing Flink for stream processing to improve the efficiency of facial ciphertext computation. This approach ensures both the security of facial data and high efficiency and practicality in the process.

2 Related Work

2.1 Homomorphic Encryption

Homomorphic encryption is one of the important techniques for privacy protection. The idea behind homomorphic encryption is to perform encryption first and then perform encrypted calculations based on the ciphertext data. The result of the encrypted computation, when decrypted, will match the result of the same computation on the original unencrypted data.

In 2017, Cheon et al. proposed the CKKS (Cheon-Kim-Kim-Song) homomorphic encryption scheme [1]. It is an approximate homomorphic encryption scheme that supports floating-point number calculations and has high efficiency. Currently, it is one of the most widely used homomorphic encryption schemes [2].

As the theory of homomorphic encryption continues to evolve, this technology has been widely applied in various fields [2–6], such as machine learning [7,8], secure multi-party computation [9–11], federated learning [12,13], and cloud computing [14–16]. However, the high computational overhead and low efficiency of homomorphic encryption privacy protection schemes have been major bottlenecks restricting the development of homomorphic encryption technology. In recent years, many experts and scholars have been dedicated to improving the efficiency

of homomorphic encryption technology [17–21]. Parallel computing techniques have been widely used to enhance the execution efficiency of homomorphic encryption schemes, such as ciphertext packing techniques [22], batch processing, and single instruction multiple data (SIMD) techniques, among others.

2.2 Flink Stream Processing Engine

Apache Flink is a distributed processing engine primarily designed to handle streaming data and supports stateful computations. In the Flink framework, all data is treated as data streams, where batch data represents bounded data streams and real-time data means unbounded data streams. As a result, Flink is a unified big data processing engine that can perform stream and batch processing operations. One of the critical features of Flink is that it is event-driven. It can receive data from multiple data sources as data streams and triggers corresponding data operations only when data is received. No procedures are performed when data is not available.

2.3 FaceRecognition

FaceRecognition is an open-source face recognition library based on the Python programming language. It utilizes a simple API to perform face detection and recognition. The library offers a range of powerful functionalities, including face alignment, feature extraction, and more, making it suitable for various applications such as face recognition and face verification.

3 Methodology

3.1 Methodology Overview

In this paper, the features extracted from facial images are first encrypted and stored in a cloud-based database. During face identity verification, the features of the current face are extracted and encrypted. The feature matching phase utilizes Flink for data stream processing, which consists of two sub-stages. The Map operator calculates the differences between encrypted data streams for each sample and then squares them. The Reduce operator aggregates the results of all data streams and computes the sum. Finally, the encrypted comparison score between the two is returned.

This scheme aims to protect the private information of users stored in the template database by performing approximate homomorphic encryption for matching calculations. Simultaneously, it utilizes Flink for data stream processing to improve computational efficiency. The approach ensures both data security and efficiency, providing a practical and effective solution.

System Model: In privacy-preserving machine learning algorithms, one of the most common computations is the matrix multiplication between two matrices. Matrices can be viewed as special types of vectors, which allows us to transform the matrix multiplication into the form of a vector inner product. The vector inner product involves the multiplication of corresponding elements in two vectors followed by their summation. The element-wise multiplication process between vectors is independent of each other, making it amenable to parallelization to improve efficiency. Finally, the results of parallel computations are aggregated and summed up. The framework can be seen in Fig. 1.

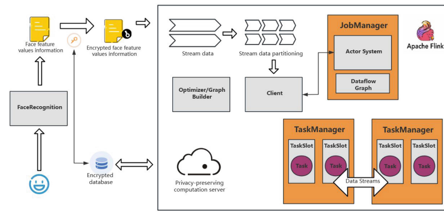

Fig. 1. Frame Work

3.2 Feature Encryption

To prevent malicious attacks on the template database, which could lead to obtaining real facial images and their corresponding feature attributes, it is necessary to protect the stored feature templates in the cloud. Therefore, the proposed scheme suggests encrypting the successfully extracted feature templates using approximate homomorphic encryption and then sending the encrypted feature templates to the cloud. The key generation and encryption process for approximate homomorphic encryption are as follows:

Key Generation: Use the SEAL library to get the locally transmitted parms build parameter container. The CKKS framework is then generated using the parameter params: contextSEALContext context(params). Obtain the key list through local transmission. After the key is generated, the decrypted private key is saved by the local client by calling the HE.save_key key saving function in the homomorphic encryption module.

The public key, relinearized key, and rotating key need to be sent to the cloud for subsequent homomorphic calculation. Relinearized key is mainly used to reduce noise. Euclidean distance calculation requires the use of rotation keys. The algorithm flow can be seen in Algorithm 1.

Feature Encryption: After obtaining the facial feature values, it is necessary to encrypt them using the generated keys. The entire facial floating-point feature values are converted into a vector called DoubleVector. The encryption

Algorithm 1: Key generation algorithm

Data: keypath
1 key_list ← HE.registration();
2 HE.save_key(keypath,key_list);
Result: public key, relinearized key, rotated key

is performed using the HE.encrypt function from the homomorphic encryption module. This function will return the encrypted ciphertext of the facial features. Subsequently, the local client uploads the encrypted facial feature ciphertext to the cloud-based ciphertext database. The algorithm flow can be seen in Algorithm 2.

Algorithm 2: Face Feature Encryption Algorithm

Data: feature
1 feature_dv ← DoubleVector(feature);
2 encrypted_feature ← HE.encrypt(feature_dv,context,publice_key);
Result: encrypted_feature

3.3 Ciphertext Feature Stream Distributed Computation

The approximate homomorphic encryption used in this paper is based on polynomial encryption, and only linear functions can perform corresponding approximate homomorphic operations. However, when the function model is nonlinear, it needs to undergo an approximate transformation to convert it into an approximate linear function.

Since face_recognition calculates the similarity between features using the Euclidean distance between them, the closer the Euclidean distance between two facial features, the higher the likelihood that the two faces belong to the same person. As shown below, the formula represents the Euclidean calculation in an N-dimensional space.

$$\mathrm{dist}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{y}_i)^2} \tag{1}$$

From the formula, it can be deduced that all operations are linear, except for the final square root calculation. In this paper, homomorphic computation is performed on the cloud side, and the homomorphic results are returned to the client, where the square root is performed in the plaintext domain.

The homomorphic encryption module currently does not directly support summation. Therefore, vector rotation is required to achieve summation with respect to the first element.

The approach proposed in this paper is based on the Flink data stream processing engine. Similar to traditional big data frameworks like Hadoop and Spark, Flink's parallel computing framework also supports the Map-Reduce paradigm. In Flink, data processing is done on data streams, where the Map operator performs operations on the data stream, and the Reduce operator aggregates the results of all data streams. Prior to the Reduce operation, a KeyBy operation is needed to split the data stream into different partitions and group data with the same key into the same partition.

In the proposed approach, we need to identify parallelizable steps in the ciphertext computation process. In the CKKS algorithm's ciphertext computation, ciphertexts are mutually independent and do not interfere with each other. Therefore, we can follow the parallelization techniques used in plaintext data processing to design parallelized computation for ciphertext data. The algorithm flow can be seen in Algorithm 3.

Algorithm 3: The Face Feature Distance Algorithm in a Data Stream Environment

```
   Data: cip1,cip_dict
 1 gal_keys ← load_key(gal_keys_path);
 2 foreach key in encrypted_feature_dict do
 3     cip_sub ← HE.sub_cipher(cip1,cip_dict[key]);
 4     cip_sqrt ← HE.square_cipher(cip_sub);
 5     cip_rot ← Ciphertext(cipher_sqrt);
 6     foreach i = 1,2,...,k do
 7         evaluator.rotate_vector(cip_rot,1,gal_key);
 8         cip_sum ← HE.add_cipher(cip_rot,cip_sqrt);
 9     end
10     score_dict[key] = cip_sum;
11 end
   Result: score_dict
```

3.4 Feature Matching

The local client receives a dictionary of encrypted squared sums of facial feature values sent from the cloud. In this dictionary, the keys represent the user information identifiers, and the values represent the encrypted squared sums of facial feature values corresponding to each user. The HE.decrypt function is used to decrypt the values, and then a plaintext square root operation is performed locally, resulting in the plaintext facial feature values. Afterward, using the corresponding plaintext feature value threshold, the client filters out user information that meets the condition of having the smallest distance between facial feature values. The algorithm flow can be seen in Algorithm 4.

Algorithm 4: Feature Matching Algorithm

```
   Data: score_dict
1  foreach key in score_dict do
2  |   score ← HE.decrypt(score_dict[key],secret_key);
3  |   score_sqrt ← Math.sqrt(score);
4  |   if score_sqrt ≤ threshold && score_sqrt ≤ score_sqrt_min then
5  |   |   score_sqrt_min = score_sqrt;
6  |   |   user_id = key;
7  |   end
8  end
   Result: user_id
```

4 Algorithm Analysis

4.1 Security Analysis

Assuming that all entities (clients and servers) in the system model are honest and curious, they can honestly perform protocol computations but may try to obtain data from other entities. An adversary is defined with the following capabilities: (1) it may eavesdrop on the transmission of facial feature data; (2) it may eavesdrop and obtain intermediate results of facial feature template matching on the server, such as the squared Euclidean distance, leading to the inference of other private data based on the intermediate results and its own facial data.

Regarding the facial feature templates, the client's information is encrypted using homomorphic encryption before uploading it to the privacy service provider. The decryption key required for decryption can only be obtained by the client, and the server cannot decrypt the encrypted feature templates. This ensures that the server cannot access users' facial information.

The squared Euclidean distance, which is an intermediate result obtained through homomorphic operations, also requires the decryption key for decryption, and thus the server cannot obtain this data. Therefore, the client's data is secure and cannot be accessed by the server or the adversary, only requiring attention to the security of approximate homomorphic encryption itself. The hardness of the RLWE problem ensures the security of approximate homomorphic encryption.

4.2 Algorithm Performance Analysis

Serial Performance Analysis

We assume that the data stream contains N data points, each consisting of m features. After encryption, there are N ciphertexts, each ciphertext containing m ciphertext slots with data. The ciphertext computations are derived from

two fundamental operations: homomorphic addition and homomorphic multiplication. As mentioned in the previous introduction to the CKKS algorithm, homomorphic addition and multiplication primarily involve additions and multiplications. In computer computations, the time consumed by addition and multiplication is roughly the same and denoted as T_m.

Homomorphic addition requires one addition operation and one modulo operation, while homomorphic multiplication requires 9 additions or multiplications and one modulo operation. Additionally, after each homomorphic multiplication, a re-scaling operation is needed, which requires 2 multiplications. The time consumed by the modulo operation is denoted as T_a.

In ciphertext computations, the time consumed by homomorphic addition or subtraction is denoted as $T_m + T_a$, and the time consumed by homomorphic multiplication is denoted as $11T_m + T_a$. Homomorphic vector dot product is implemented through operations that traverse ciphertext slots. Assuming the time taken to traverse one ciphertext slot is denoted as T_c, the time consumed by homomorphic vector dot product is $(10m + 2)T_m + T_a + mT_c$.

In this scheme, calculating the Euclidean distance requires N homomorphic vector subtractions, N homomorphic vector dot products, and N homomorphic additions. The time taken for homomorphic vector subtraction is $Nm(T_m + T_a)$, the time taken for homomorphic vector dot product is $N[(10m+2)T_m+T_a+mT_c]$, and the time taken for homomorphic addition is NT_m.

When performed sequentially, the time consumed for calculating the Euclidean distance is denoted as T_{serial}, and it is equal to the sum of the times taken for vector subtraction, vector dot product, and addition: $T_{serial} = Nm(T_m + T_a) + N[(10m + 2)T_m + T_a + mT_c] + NT_m = (11m + 3)NT_m + (m + 1)NT_a + NmT_c$.

Parallel Performance Analysis

Assuming parallelism, the data stream is divided into s sub-data streams and assigned to s maps for ciphertext computations. Each sub-data stream contains s data points, so it satisfies $k = N/s$. Assuming the cluster has d nodes, and on average, each node can process v sub-data streams, then $s = dv$.

The parallel ciphertext computation process can be divided into two parts: Map and Reduce. In the Map phase, data stream computations and communication between Maps are performed. In the Reduce phase, node scheduling and merging/sorting of sub-data streams are carried out. Let the time consumed by a single Map be M_0, and the total time for the Map phase be M. Then:

$$\begin{aligned} M &= vM_0 \\ M_0^{main} &= (11m + 3)kT_m + (m + 1)kT_a + kmT_c \end{aligned} \tag{2}$$

When there are s Maps working in the cluster, there will be at least $2s$ communication events. Let the communication time be $T_f = \delta s T_m$. The speedup of the algorithm in the Map phase can be calculated as follows:

$$\begin{aligned}\frac{T_{main}}{T_{Map}} &\approx \frac{T_{main}}{vM_0^{main}+T_f}\\ &= \frac{(11m+3)NT_m+(m+1)NT_a+NmT_c}{v[(11m+3)kT_m+(m+1)kT_a+kmT_c]+\delta sT_m}\\ &= \frac{1}{\frac{kv}{N}+\frac{\delta sT_m}{(11m+3)NT_m+(m+1)NT_a+NmT_c}}\\ &= \frac{d}{1+\frac{d\delta ST_m}{(11m+3)NT_m+(m+1)NT_a+NmT_c}} \approx d\end{aligned} \tag{3}$$

In practical environments, N is usually much larger than the number of cluster nodes and the number of sub-data streams, and the communication time between nodes is almost negligible. Therefore, the theoretical speedup in the Map phase can reach the theoretical value of d.

In the Reduce phase, the main time-consuming tasks are the integration and sorting of computation results from Map nodes and communication among nodes. Let the communication time in this process be $T_{fr} = \delta_1 sT_m$. Additionally, using a sorting algorithm with a complexity of $O(slogs)$, the sorting time is denoted as $S = \varepsilon slogsT_m$.

The overall speedup in the parallel computation process can be calculated as follows:

$$\begin{aligned}\frac{T_{main}}{T_P} = \frac{T_{main}}{vM_0^{main}+T_f+S+T_{fr}} &\approx \frac{T_{main}}{vM_0^{main}+T_f}\\ &\approx \frac{T_{main}}{T_{Map}} \approx d\end{aligned} \tag{4}$$

In the experimental environment of this paper, the average computation time for the Map phase is 0.5121 s, while the average computation time for the Reduce phase is 0.6163 s. The Reduce phase's computation time is 1.204 times that of the Map phase.

$$S + T_{fr} \approx 1.2(vM_0^{main}+T_f) \tag{5}$$

The overall speedup is given by:

$$\begin{aligned}\frac{T_{main}}{T_P} &= \frac{T_{main}}{vM_0^{main}+T_f+S+T_{fr}}\\ \approx \frac{T_{main}}{vM_0^{main}+T_f+1.2(vM_0^{main}+T_f)} &\approx \frac{1}{2.2}d\end{aligned} \tag{6}$$

Based on the above analysis, it can be concluded that due to the relatively large computation time in the Reduce phase, the overall speedup will be approximately 1/2.2 times the numwber of cluster nodes.

5 Experiments

Precision Comparison: In this experiment, the Labeled Faces in the Wild (LFW) dataset was used, which consists of 5,749 folders containing 13,233 images. The face verification task was performed using the face pairs information provided in pairs.txt, which includes 3,000 matching pairs and 3,000 non-matching pairs. The goal was to determine the facial similarity and verify whether two face images belonged to the same person.The CPU device used in this experiment is: 11th Gen Intel Core i7-11800H @ 2.30 GHz, Octa-Core. The memory device used is: 32 GB (Kingston DDR4 3200MHz). The Flink cluster was set up using Ubuntu virtual machines, consisting of one Taskmanager and two Jobmanagers.

The results are shown in Fig. 2.

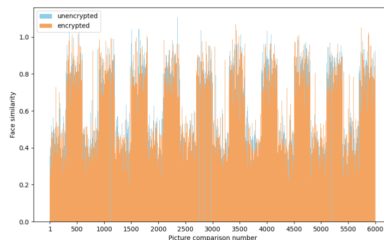

Fig. 2. Face Similarity

Obtain the matching results and conduct error analysis of the similarity between the unencrypted and encrypted schemes. The calculation of the error rate for the encryption scheme is done using the following formula:

$$error_rate = \frac{ABS(Unencrypted - Encrypted)}{Unencrypted} * 100 \tag{7}$$

Figure 3 corresponds to the error rate for 6,000 pairs of face verification, and the final accuracy was 98.05%, with an average error rate of 0.004618455%. The face verification scheme based on homomorphic encryption produces similarity calculation values that are close to the values obtained in plaintext. Furthermore, when compared with the accuracy results from [23], as shown in the Table 1, our method significantly improves accuracy and demonstrates higher efficiency.

Further experimental comparison of unencrypted and encrypted data is shown in Fig. 4. From the ROC curves of the False Acceptance Rate (FAR) and True Acceptance Rate (TAR), it can be observed that both unencrypted and encrypted ROC values are 98.3401%. The parallel processing of homomorphic encryption hardly affects the recognition accuracy, and this experimental result is consistent with the error analysis results.

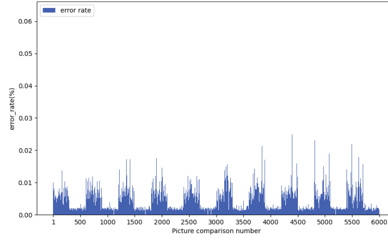

Fig. 3. Face Feature Error Rate

Table 1. Accuracy Comparison

Method	[24]	[25]	[23]	[ours]
Accuracy of the top-1 match	91.55%	95.50%	96.60%	98.05%
Feature volume	1600bits	3000bits	256bits	128bits

The smaller the values of False Acceptance Rate (FAR) and False Rejection Rate (FRR), the better the performance. However, changes in individual metrics can affect other metrics. From the Detection Error Tradeoff (DET) curve of FAR and FRR, it can be observed that the encrypted ERR (Equal Error Rate) value is 0.02567. A curve that leans towards the lower-left corner indicates better performance of the scheme, meaning that the difference between actual and measured values is small. The results are shown in Fig. 5.

Efficiency Comparison: This paper conducts serial and parallel tests on encrypted face recognition using a test dataset containing 10,000 pairs of encrypted face images. The parallelism of the cluster is adjusted by controlling the available slot slots and task parallelism. The experiment records the training time of the algorithm in serial and with different degrees of parallelism in the cluster. By using the serial time as a reference, the speedup ratio for different degrees of parallelism is calculated, and finally, the experimental results are analyzed. The results are shown in Table 2 and Fig. 6.

Based on the analysis of Table 2 and Fig. 6, the following conclusions can be drawn regarding the increase in available slots and parallelism of the Flink cluster:

(1) The overall execution time of the ciphertext gradually decreases with the increase in parallelism. Particularly, before reaching the maximum available slots in the cluster, the execution time of the ciphertext significantly decreases, leading to a notable improvement in algorithm performance. Once the parallelism exceeds the maximum available slots in the cluster, all nodes are fully utilized for computation. Moreover, the impact of increased parallelism on the cluster performance becomes relatively small, and the ciphertext execution time stabilizes gradually.

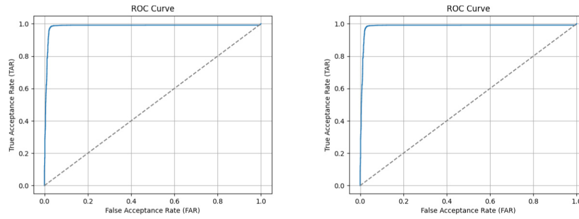

Fig. 4. ROC Curve for Encrypted and Unencrypted Data

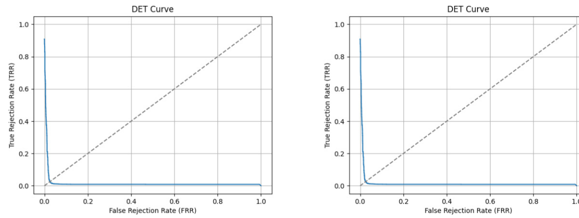

Fig. 5. DET Curve for Encrypted and Unencrypted Data

(2) As the size of the Flink cluster increases, theoretically, it can handle more data volume and more complex computational tasks. However, in practical applications, the extent of efficiency improvement does not always directly correlate with the size of the cluster. This is mainly due to the following factors:
  - Data Partitioning: For some datasets, it may be challenging to distribute them evenly among all nodes in the Flink cluster. This leads to some nodes being overloaded while others remain idle, which decreases the overall efficiency of the cluster.
  - Network Communication: With the growth of the cluster, the amount of data that needs to be transferred between nodes also increases. If the network bandwidth is insufficient or there are high network latencies, it can slow down the communication between nodes, thus affecting the overall efficiency of the cluster.
  - Hardware Limitations: Before increasing the size of the Flink cluster, it is essential to ensure that each node has sufficient hardware resources such as memory, CPU, and disk space. Otherwise, adding more nodes may result in wastage of resources without a corresponding increase in efficiency.

(3) Due to the significant time consumption in the Reduce stage, the overall speedup ratio decreases. The highest overall speedup ratio is 4.8, which is close to $9/2.2 = 4.09$, and it is consistent with the theoretical analysis value mentioned earlier. Moreover, compared [23], the average encryption matching time per pair of images has reduced from 0.71 s to 63.5 ms, greatly improving the efficiency.

Table 2. Runtime and Speedup at Different Parallelism Levels

Parallelism	Execution time/ms	Speedup
1	3032227	1.00
2	1581397	1.91
3	1081455	2.80
4	898669	3.37
5	751976	4.03
6	690305	4.39
7	656067	4.62
8	643639	4.71
9	635354	4.77
10	632041	4.80
11	632626	4.79
12	636156	4.77

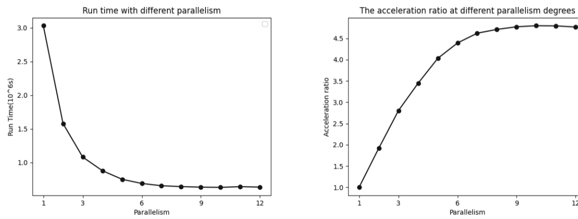

Fig. 6. Runtime and Speedup Curves

Feasibility Analysis: Based on the precision comparison experiments conducted on encrypted data, it can be concluded that the average error rate of the face recognition algorithm in the current encrypted domain is approximately 0.004618455%, which ensures high accuracy in practical application scenarios.

Furthermore, based on the efficiency comparison experiments, in the experimental environment with nine slots, the highest parallel efficiency for retrieving 10,000 facial images took approximately 635,354 ms. The average retrieval time for each image in the database was 63.5 ms. According to the conclusions from the performance analysis of parallel algorithms, by increasing the slots to 54, the acceleration ratio can reach approximately 24.5 times, and the retrieval time for each image is around 10.58 ms. If the encrypted clustering search algorithm is also applied simultaneously, it can effectively meet the requirements for daily applications.

6 Conclusion

This paper proposes a parallel privacy-preserving homomorphic encryption face verification scheme to address the issues of face information leakage and low computational efficiency in traditional face recognition. The scheme reconstructs the feature template matching protocol based on the homomorphic encryption module to achieve feature vector matching in the ciphertext domain. Security analysis and experimental comparisons demonstrate that the proposed scheme can achieve vector feature template matching while ensuring the security of the feature templates.

Leveraging the parallel computing concept of Map-Reduce, the scheme combines the stream data processing engine Flink with the homomorphic encryption algorithm CKKS to accelerate feature value matching in the ciphertext domain. Through algorithm performance analysis and experimental results, the acceleration ratio and accuracy of the algorithm are verified.

In conclusion, the proposed approach achieves efficient and accurate face verification while ensuring the security of data information during transmission and storage in the cloud.

References

1. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
2. Titus, A.J., Kishore, S., Stavish, T., Rogers, S.M., Ni, K.: Pyseal: a python wrapper implementation of the seal homomorphic encryption library (2018)
3. Li, Y., Ng, K.S., Purcell, M.: A tutorial introduction to lattice-based cryptography and homomorphic encryption (2022)
4. Dowerah, U., Krishnaswamy, S.: Towards an efficient LWE-based fully homomorphic encryption scheme. IET Inf. Secur. **16**(4), 16 (2022)
5. Xiaoming D., Department, E.T.: Research on fully homomorphic encryption schems. Electronics World (2016)
6. Zhang, Z., Cheng, P., Chen, J., Wu, J.: Secure state estimation using hybrid homomorphic encryption scheme. IEEE Trans. Control Syst. Technol. **29**, 1704–1720 (2020)
7. Wang, Y., Liang, X., Hei, X., Ji, W., Zhu, L.: Deep learning data privacy protection based on homomorphic encryption in aiot. Mob. Inf. Syst. **2021**(2), 1–11 (2021)
8. Park, J., Kim, D.S., Lim, H.: Privacy-preserving reinforcement learning using homomorphic encryption in cloud computing infrastructures. IEEE Access **8**, 203564–203579 (2020)
9. Aloufi, A., Hu, P., Song, Y., Lauter, K.: Computing blindfolded on data homomorphically encrypted under multiple keys: a survey. ACM Comput. Surv. (CSUR) **54**, 1–37 (2021)
10. Yang, X., Yi, X., Kelarev, A., Han, F., Luo, J.: A distributed networked system for secure publicly verifiable self-tallying online voting. Inf. Sci. **543**, 125–142 (2021)
11. Xu, W., Wang, B., Hu, Y., Duan, P., Zhang, B., Liu, M.: Multi-key fully homomorphic encryption from additive homomorphism. Comput. J. **66**(1), 197–207 (2023)

12. Fang, H., Qian, Q., Chen, M.L.: Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet **13**, 94 (2021)
13. Wibawa, F., Ozgur Catak, F., Sarp, S., Kuzlu, M., Cali, U.: Homomorphic encryption and federated learning based privacy-preserving cnn training: Covid-19 detection use-case. arXiv e-prints (2022)
14. Zhang, J., Jiang, Z.L., Li, P., Yiu, S.M.: Privacy-preserving multikey computing framework for encrypted data in the cloud. Inf. Sci. **575**, 217–230 (2021)
15. Park, J.H.: Homomorphic encryption based privacy-preservation for iomt. Appl. Sci. **11**, 8757 (2021)
16. Mohammed, S., Basheer, D.: From cloud computing security towards homomorphic encryption: a comprehensive review. TELKOMNIKA (Telecommunication Computing Electronics and Control) (2021)
17. Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 1–16. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30057-8_1
18. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 129–148. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_9
19. Tibouchi, M.: Fully homomorphic encryption over the integers: from theory to practice. NTT Techn. Rev. **12**(7), 273–81 (2014)
20. Zhao, D.: Rache: radix-additive caching for homomorphic encryption (2022)
21. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
22. Ertaul, L.: Implementation of homomorphic encryption schemes for secure packet forwarding in mobile ad hoc networks (manets) (2022)
23. Ma, Y., Wu, L., Gu, X., He, J., Yang, Z.: A secure face-verification scheme based on homomorphic encryption and deep neural networks. IEEE Access **5**, 16532–16538 (2017)
24. Jin, X., Liu, Y., Li, X., Zhao, G., Guo, K.: Privacy preserving face identification in the cloud through sparse representation. In: Chinese Conference on Biometric Recognition (2015)
25. Osadchy, M., Pinkas, B., Jarrous, A., Moskovich, B.: Scifi - a system for secure face identification. In: IEEE Symposium on Security & Privacy (2010)