Neural Processing Letters
https://doi.org/10.1007/s11063-021-10477-y

# Privacy Enhanced Cloud-Based Facial Recognition

**Tao Yang[1] · Yuhang Zhang[1] · Jie Sun[2] · Xun Wang[2]**

Accepted: 26 February 2021


**Abstract**
Homomorphic encryption is a significant method to protect user privacy in cloud computing environment. Due to the computation efficiency issue, there is still not many homomorphic encryption applications for common users. In this paper,we try to use homomorphic encryption to enhance the privacy in cloud-based face recognition system. By balancing the workload between client and server,and reimplementing the similarity measurement function, our homomorphic encryption version's performance is almost the same as the original version in terms of accuracy and time consumption. Our work is especially beneficial to many face recognition methods that are using Euclidian distance as their similarity metric.

**Keywords** Homomorphic encryption · Face recognition

## 1 Introduction

Cloud based face recognition has been more and more used in many areas. Leveraging the remote computation resources, personal computers and mobile devices can complete heavy workload which usually needs a large scale of computation, such as face recognition, object detection, and object recognition. However, human face images contain sensitive personal information,such as identification informance, health situation, relationship between others etc. When face images are sent into a third party cloud service provider,they are also vulnerable to information hijacking in information transmission or privacy leakage in the untrusted third party cloud service provider. As a result, researchers are looking for the approaches to leverage untrusted third party computation resources and not leak users' privacy in the meanwhile.

Generally,there are two ways to protect user privacy in cloud computing scenarios. One is the secure multi-party computation(SMC) [1,28]. SMC creates methods for parties to jointly compute a function over their inputs while keeping those inputs private. SMC needs the client and server work interactively, which is not the common paradigm in our daily applications. The other one is using Homomorphic Encryption(HE). HE is also known to be an effective

✉ Jie Sun
sunjie@zjgsu.edu.cn

[1] Department of Information Security, Zhejiang Gongshang University, Hangzhou, China

[2] Department of Computer Science, Zhejiang Gongshang University, Hangzhou, China

Published online: 13 March 2021

approach to protect the user privacy in third party cloud service providers. Users' data is encrypted with a public key of the homomorphic encryption(HE) scheme and uploaded to the third-party cloud storage. In homomorphic encryption, users' data is processed within encryption domain. The results after processing are also sent back to users in homomorphic ciphertext. Users only decrypt the results with their private keys without revealing their data to the cloud service providers. In HE, not only the data, but also the computation operations are also encrypted. Ideally, the operation and the data can be added and multipled in arbitrary numbers of time, which is so called Fully Homomorphic Encryption(FHE) [12]. However, FHE is hard to widely used in reality because of its large computation overhead. To reduce the computation complexity, people are thinking to give up the arbitrary time of multiplication and constrain the homomorphic computation within certain scale. Within that scheme, only limited number of multiplication are allowd. That scheme is so called Somewhat Homomorphic Encryption(SHE) [2,18,23].

Homomorphic encryption has also been used in biometric research applied homomorphic encryption in finger print recognition [9]. Use FV scheme as their encryption scheme [19]. The work we presented in this paper is focusing on the face recognition application for cloud computing scenarios. To protect user privacy in the cloud, our solution combines SHE with current wide-use face recognition model. We believe even with a limit number of multiplication, it is possible to convert a pretrained facial recognition model into a privacy secured homomorphic-friendly model, in the meanwhile keep the same recognition rate and acceptable processing speed. There are two contributions of this paper. First, it provides a privacy secured solution for cloud based face recognition application via homomorphic encryption. Second, this paper proposes a converting method that can be used in other face recognition methods that use eulidian distance as the similarity metrics.

The layout of the paper is like following: Sect. 2 introduces the background of homomorphic encryption schemes, the homomorphic encryption framework we used in our system, and the introduction of CKKS scheme. Section 3 explains some problems in system architecture designing. Section 4 shows the performance evaluation in terms of accuracy and computation efficiency.The discussion about the results and future work is in Sect. 5.

## 2 Related Background

### 2.1 Somewhat Homomorphic Encryption Schemes

After Gentry *et al* firstly proposed Full Homomorphic Encryption (FHE) [12] in 2009, other full homomorphic encryption methods are proposed to improve the computation efficiency [27]. But large size of keys and low computation efficiency still make FHE difficult to be widely used. They were soon replaced by the Ring Learning With Errors(RLWE).This kind of cryptographic schemes are dependent on the hardness of the Ring Learning With Errors (RLWE) problem. There are three most efficient homomorphic encryption schemes Brakerski-Gentry-Vaikuntanathan (BGV) [4], Brakerski/Fan-Vercauteren (BFV) [3,10] and the Cheon-Kim-Kim-Song (CKKS) [6]. In BGV and BFV, computations can only be performed on integers. while in CKKS, computations can be performed on complex numbers with limited precision. As CKKS can process float numbers, we choose the CKKS scheme as our homomorphic encryption scheme.

### 2.2 Homomorphic Encryption Library Framework

To implement a practical homomorphic encryption system, many homomorphic encryption library frameworks have been developed to help implementing the homomorphic encryption scheme. Those frameworks implement addition, multiplication and other computation operations and provide the wrapped functions as APIs to common users, which can greatly accelerate the implementation. SEAL [22] is developed by the Microsoft Research Cryptography Research Group. It is originally developed in C++, and only provides BFV and CKKS schemes. HElib [13] is developed by Shai Halevi and Victor Shoup. It has longer history than SEAL, but has less community support. PySEAL [26] is a wrapper implementation to the original SEAL Library.It provides user APIs in python language. In this paper, we choose the pyseal to be our homomorphic encryption library.

### 2.3 CKKS Encryption Scheme

CKKS scheme is a homomorphic scheme for approximate arithmetic. It supports an approximate addition and multiplication of encrypted messages,together with a new *rescaling* procedure for managing the magnitude of plaintext.The main idea is to add a noise following significant figures which contain a main message. This noise is originally added to the plaintext for security, but considered to be a part of error occurring during approximate computations that is reduced along with plaintext by rescaling [6].

The main idea of CKKS is to treat an encryption noise as part of error occurring during approximate computations. If $C$ is the ciphertext for the message $m$ by the private key $sk$, the decryption form is

$$\langle c, sk \rangle = m + e \, (mod \, q) \tag{1}$$

where $e$ is a small error inserted to guarantee the security of hardness assumptions such as the learning with errors(LWE), the ring-LWE(RLWE).If $e$ is relatively small enough to the message $m$, the noise will not affect the message and the value $m' = m + e$ can replace the original message in approximate arithmetic.

To treat the encryption noise as precision loss, the CKKS has to maintain the result of multiplication at a certain level. The *rescaling* technology is used to maintain the level of product and makes CKKS possible to process the small-degree polynomial, which is a common case for float and complex numbers. For example, the result of the Eq. 1 after rescaling is $\lfloor p^{-1} \cdot c \rceil (mod \, q/p)$.

## 3 System Architecture

To create a privacy secured face recognition in cloud, there are two issues need to be considered. One is what kind of facial recognition method is suitable to convert to homomorphic friendly method and also maintain the recognition accuracy. The other one is, in the load balancing perspective, how to divide the workload between client and server.

### 3.1 Facial Recognition Model

People have developed many face recognition methods, some methods are using Elastic Graph Dynamic Link Model [16,17], some methods are using Gabor-filter features combined with

svm [7,11]. In the past ten years, the development of convolutional neural networks have been used in face recognition and brought the accuracy of recognition rate to a new level [21,24,25].

As the intrinsic characteristic of homomorphic encryption, two requirements for the facial recognition method need to be satisfied before it can be converted to privacy-secured homomorphic-friendly method: (1) since CKKS scheme is based on polynomial encryption, only linear functions can be encrypted by homomorphic encryption scheme, non-linear functions in the model have to make an approximation or change to another linear function. (2) the computation complexity and precision should be controlled at a certain level, large computation complexity will affect the usability of the system.

The method we presented in this paper is especially useful to the methods that determine the results by comparing the euclidian distance between two feature embeddings, such FaceNet [21], Deep Face Recognition [5,20]. Equation 2 is the equation that computes the euclidian distance between two embeddings. Since most operations in this equation are linear operation(except square root computation), it is suitable to convert it homomorphic encryption function.

$$d(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (2)$$

We used the face_recognition package in python pip repository. Python face_recognition package is implemented based on *dlib* library [8]. The model used by dlib face recognition is *dlib_face_recognition_resnet_model_v1* model, which is a resnet neural network [14] trained by 3 million human face images.

### 3.2 Workload Balancing

Another important issue need to address is how to divide the workload between the client and the server in the cloud. The motivation to use the computation resources in the cloud is moving a part of workload to the cloud. Leveraging the computation resources in cloud, even common computers and mobile devices is capable to complete some computation-intensive tasks.However, moving workloads to the cloud in homomorphic scenario does not mean reduction in computation complexity. The overhead of homomorphic encryption and decryption may affect the usability of the application.

A typical face recognition application has four parts, shown in Fig. 1, *face detection*, *face alignment*, *face feature extraction*, and *matching*. For both verification and identification tasks in face recognition, the major efficiency bottleneck is the *matching* step. In our application, *matching* is moved to the cloud.

## 4 Implementation Details and Evaluation

### 4.1 Image Encoding

The process of image encoding in our application is computing the embedding of the input image. Using the pretrained model dlib_face_recognition_resnet_model_v1, the landmark of input face image will be extracted and will get a feature vector which contains 128 floating numbers.

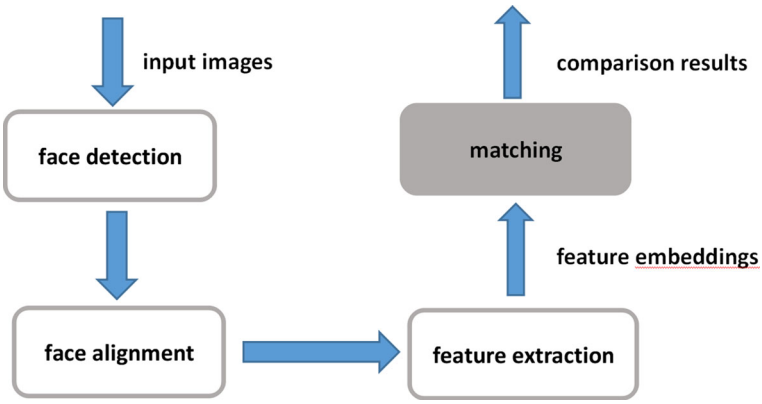



**Fig. 1** The layout of the components in a typical face recognition system. The *matching* resides in the cloud

### 4.2 Initialization

According to the CKKS scheme, every multiplication in ciphertext will make the level of the ciphertext increase one. Because the ciphertext in different levels can not be computed. The $relin\_key$ is introduced to reduce the level of the ciphertext. This process is so called "relinearization". $gal\_key$ is the galios key that will be used in rotating the cipher vector.

In our preliminary test, the size of the aformentioned keys is more than 70M bytes, which makes it is impractical to upload the keys every time with the embedding information. As a result, an initial step is necessary and once the client establishes the connection with the server, those two keys are sent to the Cloud. These two keys are not related to information encryption, the privacy of facial images will not be affected. Figure 2 shows the data communication between the client and the cloud.

### 4.3 Square Root

As we mentioned in the previous sections, only linear functions can be transferred into homomorphic style functions, square root function is difficult to implement. In our implement, we firstly choose to compute $\sum_{i=1}^{n}(x_i - y_i)^2$, send the result back to the client, and compute the square root in plaintext domain.

One thing need to be point out, in Eq. 2, those variables $x_1$,$x_2$, $x_i$ are all encoded by CKKS into polynomials. The steps of implementation is like following:

1. function *sub_inplace(cipher_embedding1-ciphter_embedding2)* is used to process $x_i - y_i$.
2. user function *square_inplace(diff_cipher_embedding)* to compute the square the $x_i - y_i$. Every time the ciphertext multiples, the level of the ciphertext increases one. Reli_keys is used in "Relinearization" to reduce the level the ciphertext.
3. Since there is no function in SEAL to sum up all the elements in one vector, we used a rotate function *rotate_vector_inplace() to rotate the vector 128 times,and sum up the first element.Then return the first element which is the* $\sum_{i=1}^{n}(x_i - y_i)^2$. The code is shown in Fig. 3.

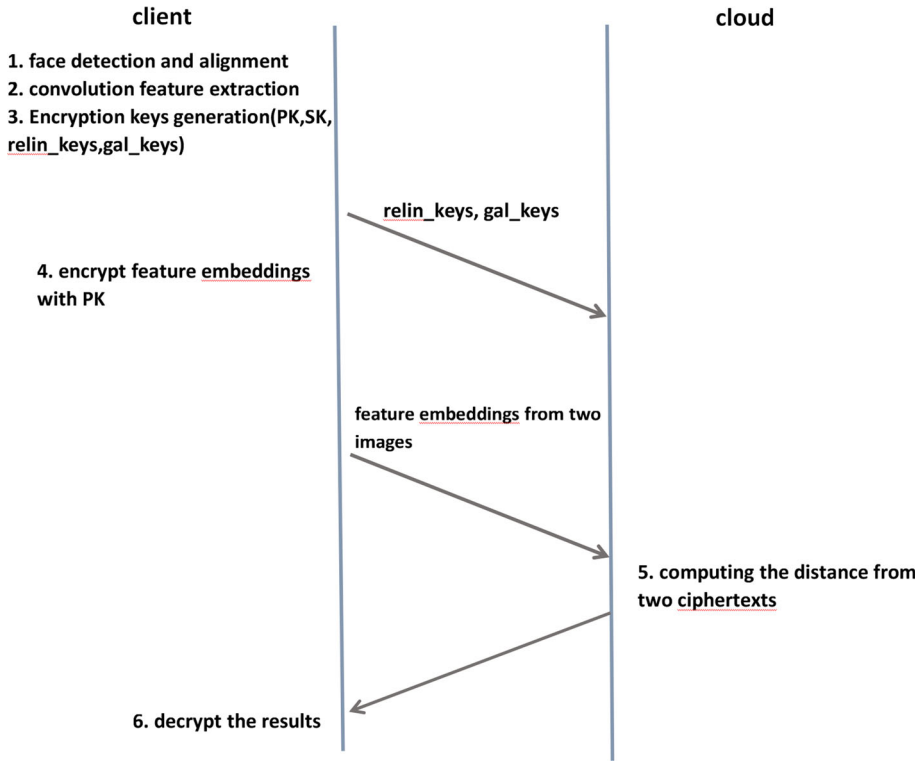



**Fig. 2** The data communication between the client and the cloud. Most components reside in client side, except the matching

```
for i in range(1,128):
    evaluator.rotate_vector_inplace(encrypted_temp, 1, gal_keys)
    evaluator.add_inplace(encrypted1,encrypted_temp)
```

**Fig. 3** Vector rotation for elements accumulation

### 4.4 Performance Test

Face recognition has two types of use cases, verification and identification. In our experiment, we only tested the cases of the verification cases. We tried to verify whether two face images belong to the same person. While the identification cases need to compare multiple face images in the dataset in order to identify the input image is one of the them or not. Our verification case can be easily extended to identification cases. It will only need to upload the embeddings and store them in the database on the cloud storage.

In the accuracy test, we use the Eq. 3 to compute the error rate. *homomorphic_score* indicates the similarity score gained from comparison two image embeddings in the homomorphic encryption scenario.

We used pairs.txt comparison list which contains 6000 comparisons in LFW dataset [15]. In Fig. 4, we can see majority error is below 0.02% which is a very low percentage. It means our implementation has equivalent accuracy as the original method. The further performance results are shown in Figs. 5 and 6. Figures 5 and 6 show both original version and homo-

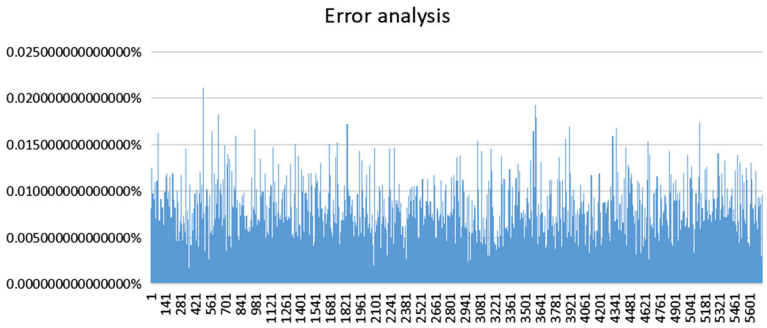
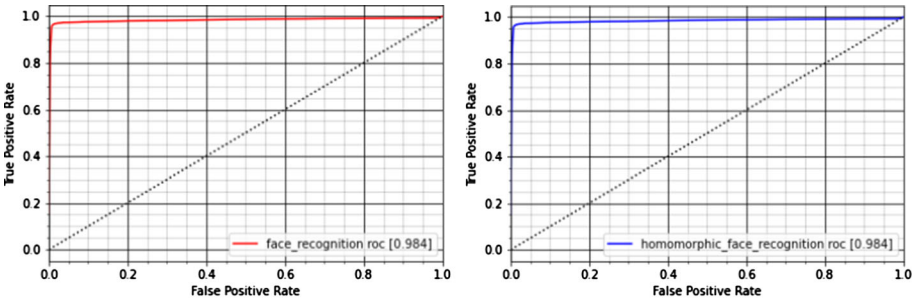
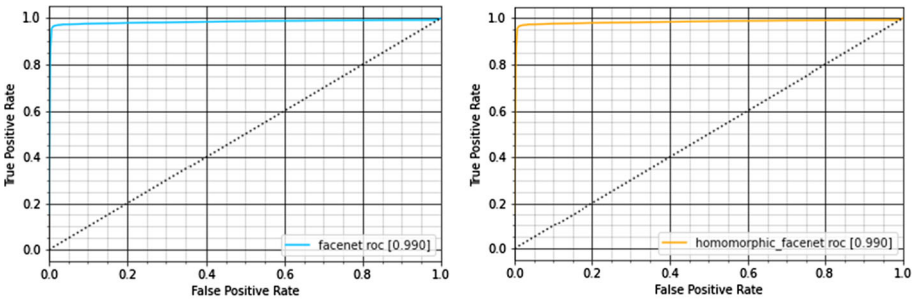



**Fig. 4** The error rate comparing the original similarity score and the score in homomorphic encryption scenario




**(a) face_recognition** **(b) homomorphic face_recognition**

**Fig. 5** ROC curve comparison between the original face_recognition method and our homomorphic modification




**(a) original facenet** **(b) homomorphic facenet**

**Fig. 6** ROC curve comparison between the original facenet and our homomorphic modification

morphic version have the same accuracy value, which means the homomorphic modification almost does not bring performance degradation in accuracy.

$$error\_rate = \frac{abs(original\_score - homomorphic\_score)}{original\_score} \times 100\% \tag{3}$$

**Table 1** Time consumption for the implementation

| Encoding | Decoding | Encryption | Decryption | Matching |
|---|---|---|---|---|
| 10 ms | 13 ms | 436 ms | 3 ms | 2112 ms |

**Table 2** Space consumption for the implementation

| Type | Encrypted embedding | Matching result |
|---|---|---|
| Memory footprint | 56 bytes | 56 bytes |

The time comlexity of our implementation is also plausible for practical use. Table 1 shows the time consumption of each step in our implementation. The matching step in cloud costs only approximate 2 s.

The space consumption of our implementation is like following. The actual memory consumption could be varied by the version of python and the CPU achitecture(32-bit or 64-bit). In our experiment, we used python 3.7 and 64 bit CPU. In Table 2, *encrypted embedding* is the ciphertext that client send to the server, *matching result* is the ciphertext that server send to the client. Both ciphtertexts are 56 bytes which is plausible for a commodity application.

## 5 Discussion and Future work

In this paper, we present an implementation of a privacy-secured web based face recognition system using CKKS homomorphic encryption scheme. By reimplementing the euclidian distance methods, the matching step could be computed using CKKS homomorphic scheme. Compare to the original method, the homomorphic encryption version has the same accuracy. And the time and space complexity are also acceptable. It is a good example to apply homomorphic encryption in face recognition area, and especially useful for the methods that use euclidian distance as their loss functions.

**Acknowledgements** The research is supported by National Key Research and Development Plan (2018YFB1404102),National Nature Science Foundation of China(U1609215),and Nature Science Foundation of Zhejiang Province(LQ20F020008).

## References

1. Barni M, Orlandi C, Piva A (2006) A privacy-preserving protocol for neural-network-based computation. In: Proceedings of the 8th workshop on Multimedia and security, pp 146–151
2. Bos JW, Lauter K, Loftus J, Naehrig M (2013) Improved security for a ring-based fully homomorphic encryption scheme. IMA international conference on cryptography and coding. Springer, Berlin, pp 45–64
3. Brakerski Z (2012) Fully homomorphic encryption without modulus switching from classical gapsvp. Annual cryptology conference. Springer, Berlin, pp 868–886
4. Brakerski Z, Gentry C, Vaikuntanathan V (2014) (Leveled) fully homomorphic encryption without bootstrapping. ACM Trans Comput Theory (TOCT) 6(3):1–36
5. Cao Q, Shen L, Xie W, Parkhi OM, Zisserman A (2018) Vggface2: A dataset for recognising faces across pose and age. In: 2018 13th IEEE international conference on automatic face and gesture recognition (FG 2018), pp 67–74. IEEE
6. Cheon JH, Kim A, Kim M, Song Y (2017) Homomorphic encryption for arithmetic of approximate numbers. International conference on the theory and application of cryptology and information security. Springer, Berlin, pp 409–437

7. Dadi HS, Pillutla GM (2016) Improved face recognition rate using hog features and svm classifier. IOSR J Electr Commun Eng 11(4):34–44
8. dlib c++ library. http://dlib.net (2020)
9. Failla P, Barni M, Catalano D, Raimondo MD, Bianchi T (2010) A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In: IEEE international conference on biometrics: theory applications systems
10. Fan J, Vercauteren F (2012) Somewhat practical fully homomorphic encryption. IACR Cryptol ePrint Arch 2012:144
11. Faruqe MO, Hasan MAM (2009) Face recognition using pca and svm. In: 2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, pp 97–101. IEEE
12. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing, pp 169–178
13. Halevi S, Shoup V (2014) Helib-an implementation of homomorphic encryption. Cryptology ePrint Archive, Report 2014/039
14. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
15. Huang GB, Ramesh M, Berg T, Learned-Miller E (2007) Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Tech. Rep. 07-49, University of Massachusetts, Amherst
16. Kotropoulos C, Tefas A, Pitas I (2000) Frontal face authentication using morphological elastic graph matching. IEEE Trans Image Process 9(4):555–560
17. Lades M, Vorbruggen JC, Buhmann J, Lange J, Von Der Malsburg C, Wurtz RP, Konen W (1993) Distortion invariant object recognition in the dynamic link architecture. IEEE Trans Comput 42(3):300–311
18. López-Alt A, Tromer E, Vaikuntanathan V (2012) On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing, pp 1219–1234
19. Morampudi MK, Prasad MVNK, Raju USN (2020) Privacy-preserving iris authentication using fully homomorphic encryption. Multim Tools Appl 79(2)
20. Parkhi OM, Vedaldi A, Zisserman A (2020) Deep face recognition
21. Schroff F, Kalenichenko D, Philbin J (2015) Facenet: A unified embedding for face recognition and clustering. Proceedings of the IEEE conference on computer vision and pattern recognition, pp 815–823
22. Microsoft SEAL, (release 3.5). https://github.com/Microsoft/SEAL (2020) Microsoft Research. Redmond, WA
23. Stehlé D, Steinfeld R (2011) Making ntru as secure as worst-case problems over ideal lattices. Annual international conference on the theory and applications of cryptographic techniques. Springer, Berlin, pp 27–47
24. Sun Y, Liang D, Wang X, Tang X (2015) Deepid3: Face recognition with very deep neural networks. arXiv preprint arXiv:1502.00873
25. Taigman Y, Yang M, Ranzato M, Wolf L (2014) Deepface: Closing the gap to human-level performance in face verification. Proceedings of the IEEE conference on computer vision and pattern recognition 1701–1708
26. Titus AJ, Kishore S, Stavish T, Rogers SM, Ni K (2018) Pyseal: A python wrapper implementation of the seal homomorphic encryption library. arXiv preprint arXiv:1803.01891
27. Van Dijk M, Gentry C, Halevi S, Vaikuntanathan V (2010) Fully homomorphic encryption over the integers. Annual international conference on the theory and applications of cryptographic techniques. Springer, Berlin, pp 24–43
28. Yao AC (1982) Protocols for secure computations. In: 23rd annual symposium on foundations of computer science (sfcs 1982), pp 160–164. IEEE