Blind-Touch: Homomorphic Encryption-Based Distributed Neural Network Inference for Privacy-Preserving Fingerprint Authentication

Hyunmin Choi[1,3], **Simon S. Woo**[2,3], **Hyoungshick Kim**[3]
[1]NAVER Cloud, South Korea
[2]Department of Artificial Intelligence, Sungkyunkwan University, South Korea
[3]Department of Computer Science and Engineering, Sungkyunkwan University, South Korea
hyunmin.choi@{navercorp.com, g.skku.edu}, swoo@g.skku.edu, hyuong@skku.edu

Abstract

Fingerprint authentication is a popular security mechanism for smartphones and laptops. However, its adoption in web and cloud environments has been limited due to privacy concerns over storing and processing biometric data on servers. This paper introduces Blind-Touch, a novel machine learning-based fingerprint authentication system leveraging homomorphic encryption to address these privacy concerns. Homomorphic encryption allows computations on encrypted data without decrypting. Thus, Blind-Touch can keep fingerprint data encrypted on the server while performing machine learning operations. Blind-Touch combines three strategies to efficiently utilize homomorphic encryption in machine learning: (1) It optimizes the feature vector for a distributed architecture, processing the first fully connected layer (FC-16) in plaintext on the client side and the subsequent layer (FC-1) post-encryption on the server, thereby minimizing encrypted computations; (2) It employs a homomorphic encryption-compatible data compression technique capable of handling 8,192 authentication results concurrently; and (3) It utilizes a clustered server architecture to simultaneously process authentication results, thereby enhancing scalability with increasing user numbers. Blind-Touch achieves high accuracy on two benchmark fingerprint datasets, with a 93.6% F1-score for the PolyU dataset and a 98.2% F1-score for the SOKOTO dataset. Moreover, Blind-Touch can match a fingerprint among 5,000 in about 0.65 seconds. With its privacy-focused design, high accuracy, and efficiency, Blind-Touch is a promising alternative to conventional fingerprint authentication for web and cloud applications.

Introduction

Fingerprint authentication is a biometric method that uses the unique characteristics of an individual's fingerprint. It is favored for smartphones and laptops because of its security and convenience (De Luca et al. 2015; Mare, Baker, and Gummeson 2016; Lovisotto et al. 2020; Cho et al. 2020). However, the adoption of fingerprint authentication in web and cloud environments faces challenges due to the risk of unauthorized access to sensitive biometric data on servers (Rui and Yan 2018). For example, in June 2015, the US Office of Personnel Management suffered a security breach in which over 5.6 million fingerprint records were stolen, highlighting the dangers of storing biometric data remotely (Gootman 2016).

In web and cloud environments, homomorphic encryption (HE) (Gentry 2009; Acar et al. 2018) is a promising solution for privacy-sensitive applications such as fingerprint authentication. HE allows computations on encrypted data without decryption on a server. Although many research works have adopted this for fingerprint authentication (Kim, Oh, and Kim 2020; Yang et al. 2020), they face challenges due to the significant computational overhead under HE. These challenges primarily arise from complex minutiae representations in fingerprints, which are computationally demanding and prone to variations that can affect the matching process. To overcome these limitations, Engelsma et al. (Engelsma, Cao, and Jain 2019) introduced a deep learning-based authentication technique using HE. However, even with its enhanced speed, an average authentication time of 3.4 seconds to search among 5,000 fingerprints (considering only the feature vector's encryption time) is not practical for real-world services. Implementing a conventional convolutional neural network (CNN) with HE is challenging due to the costly operations required in the convolution and pooling layers. Specifically, convolution layers necessitate rotations and multiplications for filter application, and pooling layers require a substantial number of multiplications. Additionally, encrypted feature vectors are significantly larger than plaintext feature vectors; therefore, we must do our best to reduce the size of the feature vector, which can consequently lead to a decrease in model accuracy.

To overcome this challenge, we introduce a novel privacy-preserving fingerprint authentication system, Blind-Touch. Blind-Touch employs a distributed deep learning architecture involving clients and a server (see Figure 1). Clients handle feature extraction through CNN operations on plaintext data while the server performs searching tasks to identify the most suitable fingerprint match with the encrypted feature vector. We optimize the feature vector size to 16, significantly less than the 192 features used in *DeepPrint* (Engelsma, Cao, and Jain 2019), by processing the first fully connected layer (FC-16) in plaintext on the client and the second fully connected layer (FC-1) post-encryption on the server. Next, to further enhance performance, we propose a novel compression method compatible with HE to process 8,192 authentication results concurrently. Finally, we

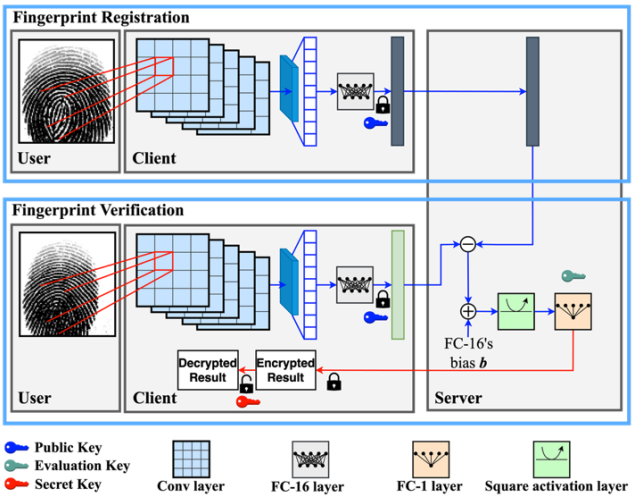

Figure 1: Overview of Blind-Touch. FC-N refers to a fully connected layer with an output size of N.

implement a clustered architecture to process authentication results simultaneously on multiple servers. To demonstrate the feasibility of Blind-Touch, we built and provided a fully functional cloud-based fingerprint authentication system (https://github.com/hm-choi/blind-touch).

Our main contributions are summarized as follows:

- **Design and implementation of Blind-Touch.** We develop a practical distributed HE-based fingerprint authentication system that facilitates efficient and precise neural network inference under HE.
- **Demonstration of Blind-Touch's superior recognition accuracy and processing time.** Blind-Touch achieves a 93.6% F1-score on the PolyU fingerprint dataset (Lin and Kumar 2018) and a 98.2% F1-score on the Sokoto fingerprint dataset (Shehu et al. 2018). Additionally, it maintains an average search time of 650 milliseconds to identify a match within a pool of 5,000 fingerprints.
- **Formal security analysis of Blind-Touch.** We provide a formal security analysis demonstrating that no probabilistic polynomial-time adversary can extract information about the encrypted fingerprint features processed by Blind-Touch in the chosen plaintext attack (IND-CPA) threat model.

Background

Homomorphic Encryption (HE)

Homomorphic encryption (HE) is an encryption scheme that allows third parties, such as cloud service providers, to perform computations on encrypted data without decryption. If m_1 and m_2 are messages, Enc denotes the homomorphic encryption function, and f and f' represent computationally feasible functions for ciphertext and plaintext inputs, then homomorphic encryption ensures $f(Enc(m_1), Enc(m_2)) = Enc(f'(m_1, m_2))$.

Researchers have introduced various homomorphic encryption algorithms over the years. For instance, Brakerski, Gentry, and Vaikuntanathan introduced the BGV algorithm (Brakerski, Gentry, and Vaikuntanathan 2014; Brakerski 2012; Fan and Vercauteren 2012), supporting integer-based arithmetic operations like addition and multiplication. The more recent CKKS algorithm (Cheon et al. 2017, 2018) supports floating point number-based arithmetic operations, making it more compatible with statistical and deep learning algorithms (Clet, Stan, and Zuber 2021). Thus, we selected the CKKS scheme for our Blind-Touch.

The CKKS method is a public-key encryption technique involving a public and a secret key. The public key includes an encryption key for encrypting floating point vectors and an evaluation key for homomorphic operations on ciphertexts. The secret key is reserved for decryption.

Siamese Neural Network with CNNs

The Siamese neural network, a deep learning structure, measures the similarity between two inputs using two identical sub-networks. Commonly used in image-matching tasks such as fingerprint authentication (Chowdhury et al. 2020) and face authentication (Song et al. 2019; Wu et al. 2017), this network generally comprises two or more CNN models as sub-networks with shared weights in the combining layer. The similarity between two input images is determined using the difference between the feature vectors computed from the sub-networks. Blind-Touch employs a Siamese neural network to compute the difference between individuals' fingerprint images.

Fingerprint Authentication with HE

HE has been regarded as an effective means for confidentially storing and processing sensitive data on the server. While prior methods using HE for fingerprint authentication (Kim, Oh, and Kim 2020; Yang et al. 2020) have employed basic filter-based models for streamlined processing, they did not achieve the accuracy performance compared to the state-of-the-art CNN-based techniques with plaintext images. Incorporating a CNN model into HE poses significant challenges due to computationally expensive operations such as convolution layers under HE.

Recent works (Dowlin et al. 2016; Folkerts, Gouert, and Tsoutsos 2021) have shown the potential of integrating HE with deep neural networks to devise both efficient and privacy-conscious machine learning models. Dowlin et al. (Dowlin et al. 2016) introduced CryptoNets, which showcases the adaptability of neural networks to encrypted data and underscores the alterations required for compatibility with HE. Similarly, Folkerts et al. (Folkerts, Gouert, and Tsoutsos 2021) proposed a framework that expanded the design of HE-driven private machine learning inference. However, their approaches fall short of accommodating CNN models with floating point parameters, which are essential for biometric verification.

Also, Engelsma et al. (Engelsma, Cao, and Jain 2019) presented DeepPrint, a deep learning-based fingerprint authentication technique that uses a fixed-length representation. DeepPrint aligns the input fingerprint, extracts a 192-dimensional texture and minutiae combination, and compresses it from floating point numbers to a 200-byte inte-

ger format. However, its 3.4-second average authentication time for 5,000 fingerprints is not viable for real-world web or cloud platforms that require quick verification. In Blind-Touch, we address such challenges by employing the CKKS scheme to accommodate a CNN with floating point parameters. We also incorporate the following three different strategies to reduce the computational and storage overheads of HE: distributed architecture, data compression, and cluster architecture. These techniques make Blind-Touch a practical solution for real-world fingerprint authentication.

Overview of Blind-Touch

We present Blind-Touch, a fingerprint authentication system that facilitates efficient inference using a Siamese neural network under HE in a distributed setup (see Figure 1). To reduce the computational load of HE, we optimized the network architecture, placing all convolutional layers on the client side, with only a fully connected layer and a square activation layer on the server side. The client captures a fingerprint and processes it using CNN operations and a fully connected layer with an output size of 16 (FC-16), then encrypts the resulting feature vector with the public key to preserve the confidentiality of the raw feature vector. We minimized the feature vector size while retaining the accuracy of fingerprint authentication, thus maximizing the number of feature vectors that can be stored in a single ciphertext. The size of 16 for the feature vector enables the storage of up to 512 fingerprint feature vectors in a single ciphertext with 8,192 slots, allowing for simultaneous comparison of an individual's fingerprint feature information against 512 registered users. The encrypted vector is then sent to the server for verification. Upon receipt, the server computes the difference between the incoming encrypted feature vector and each registered user's stored encrypted vector. The server computes the encrypted feature vector using the evaluation key. A fully connected layer with an output size of 1 (FC-1) and a square activation layer are subsequently applied to this differential input. The resulting values are relayed back to the client. Utilizing the sigmoid operation, the client decrypts the received data with the secret key and determines the highest match probability for a registered user's feature vector. By comparing this probability against a predetermined threshold, the client checks whether the provided fingerprint image matches the registered user's image. Theorem 1 demonstrates that this design yields equivalent results to processing FC-16 and FC-1 post-encryption.

Blind-Touch's sub-CNN network has a feature size of 16, derived from the FC-16 layer. Originally, the sub-CNN network had five CNN layers, with the FC-16 layer following the subtraction. In this setup, the feature vector's size could increase to 25,088. However, we discovered the FC-16 layer can be computed before encryption. Theorem 1 proves that for any linear function f defined as $f(x) = xA + b$ and any homomorphic function h, the following is true:

$$f(h(x_1) - h(x_2)) = h(f(x_1)) - h(f(x_2)) + b$$

The function f can be applied to the encrypted data before decryption. This implies that the FC-16 layer can be computed pre-encryption, reducing the feature vector's size to 1. After subtraction, the bias b of the FC-16 layer is added to the encrypted data. Consequently, Blind-Touch achieves the same accuracy as the original configuration while reducing computational cost and data encryption requirements.

Theorem 1. *Let f be a linear function defined as $f(x) = xA + b$ and h be a homomorphic function. For any x_1, x_2, and floating-point vector b, the following holds:*

$$f(h(x_1) - h(x_2)) = h(f(x_1)) - h(f(x_2)) + b$$

Proof. Starting from the left side:

$$\begin{aligned} f(h(x_1) - h(x_2)) &= (h(x_1) - h(x_2))A + b \\ &= h(x_1)A - h(x_2)A + b \\ &= h(x_1A + b) - h(x_2A + b) + b \\ &= h(f(x_1)) - h(f(x_2)) + b \end{aligned}$$

□

For training, we utilize a publicly available fingerprint image dataset. Using pairs of same-user and different-user fingerprints from this dataset, Blind-Touch can be trained to accurately identify and match fingerprints. Notably, this training is conducted using plaintext images as we employ publicly available fingerprint images, not specific individuals' private fingerprint data. Once trained, the network is repurposed for authentication of registered users.

Next, we describe fingerprint registration and authentication procedures in detail.

Key Generation and Distribution

A system administrator oversees multiple client devices with fingerprint scanners. Utilizing a key generator, the administrator generates four unique keys grouped into three types: the *public key* for encrypting fingerprint data; the Galois key for rotating ciphertext; the relinearization key for reducing ciphertext size after multiplication; and the *secret key* for data decryption. This paper terms the Galois and relinearization keys collectively as the *evaluation key*. The administrator securely embeds the public and secret keys on client devices. Subsequently, the public and evaluation keys are relayed to authentication servers via a secure channel. Administrators can then efficiently establish keys on their managed devices and register the public keys with the server, adhering to a standard key setup protocol.

Fingerprint Registration

When a user (denoted by u) registers her fingerprint with Blind-Touch, it begins by extracting and processing the feature vector of the fingerprint image using the CNN model's layers on the client side. The final layer of the CNN model on the client side is a fully connected layer (FC-16), which generates a 16-element feature vector denoted by $< u_1, u_2, \cdots, u_{16} >$. We encrypt the feature vector using the client's secret key to protect the feature vector from the server. During fingerprint registration, the client creates a ciphertext C_u containing the user's encrypted feature vector in its first 16 elements. The remaining space in the ciphertext is filled with zeros, as illustrated in Figure 2. Finally, the client

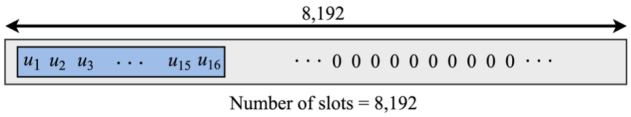

Figure 2: Ciphertext C_u containing the user u's encrypted feature vector for fingerprint registration.

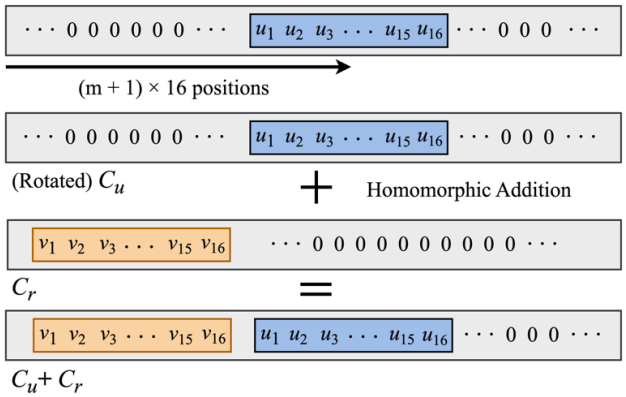

Figure 3: Addition of the rotated C_u and C_r.

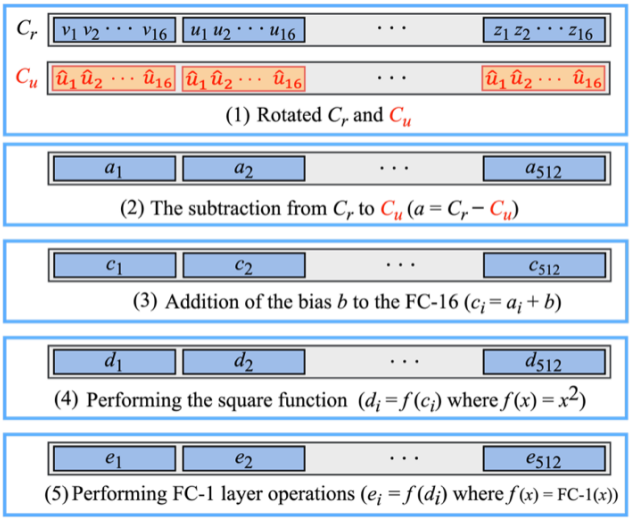

Figure 4: Server-side computations for authentication.

sends C_u to the server, along with the user's unencrypted identity information ID_u.

In Blind-Touch, the server stores encrypted feature vectors of registered users in a dedicated ciphertext, denoted as C_r. Since each feature vector has a size of 16, and C_r can hold up to 8,192 available slots, C_r can store a maximum of 512 feature vectors. When a user u registers, the server searches for 16 consecutive empty slots in C_r to store her encrypted feature vector. Assuming that the number of registered users is less than or equal to 512, the server can always find sufficient empty slots to store the user u's feature vector. Suppose m users have already registered, and u is the (m+1)th user. To store u's encrypted feature vector in C_r, the server first stores the unencrypted user identity information ID_u in a plaintext database with the index m+1. Then, the server rotates C_u by (m+1)×16 positions to the right to align it with the empty slots in C_r. This is done to ensure that the encrypted feature vectors of all registered users are stored in contiguous blocks in C_r. Finally, the rotated C_u is added to C_r, as illustrated in Figure 3.

Fingerprint Authentication

When user u attempts to authenticate using a new fingerprint image, the feature vector of this image is extracted and processed on the client side through the CNN model's layers, mirroring the fingerprint registration phase. This yields a 16-element feature vector, $< \hat{u}_1, \hat{u}_2, \cdots, \hat{u}_{16} >$, from the FC-16 layer of the CNN model. For encryption, the client produces a ciphertext C_u comprising the user's encrypted feature vector $< \hat{u}_1, \hat{u}_2, \cdots, \hat{u}_{16} >$, replicating this vector 512 times. The client then forwards C_u to the server for authentication.

The server-side computations are sequentially represented in Figure 4 as follows: (1) Upon receiving C_u, the server attempts to compare C_u with C_r; (2) the server performs the subtraction of $a = C_r - C_u$; (3) the server computes the addition of the bias b to the a; (4) the server performs the square function necessitating a multiplication; (5) the server conducts FC-1 layer operations necessitating a multiplication. Note that traditionally executing the fully connected layer operations would entail iterative rotations and additions after multiplying the result of the previous layer with the coefficients of the FC-1 layer. To avoid these multiplications, Blind-Touch repeatedly puts the identical feature vector $< \hat{u}_1, \hat{u}_2, \cdots, \hat{u}_{16} >$ into C_r. Consequently, this authentication method requires only two multiplications.

In Blind-Touch, the $sigmoid$ function is used to compute the probability of matching the feature vector of each registered user with the new fingerprint image's feature vector. However, the $sigmoid$ operation is computationally expensive with HE. To address this challenge, Blind-Touch offloads the $sigmoid$ function from the server to the client. After the server performs the fully connected layer operations, it transmits the results to the client in an encrypted form. The client then efficiently performs the $sigmoid$ operation after decrypting the results. Finally, the client compares the computed probability with a predefined threshold to determine whether the given fingerprint image matches a registered user's fingerprint image. This approach significantly reduces the computational burden on the server, making the authentication process more efficient.

Compression Method

If the number of registered users (N) on the server surpasses 512, without compression, $k(= \lceil N/512 \rceil)$ ciphertexts are necessary to relay the results for all N users because a single ciphertext can only encapsulate the authentication results for a maximum of 512 registered users. This can lead to a significant increase in the authentication time.

To address this issue, we propose a compression method, a new technique that consolidates multiple authentication results into a single ciphertext. To compress the results, the server multiplies a one-hot vector with the first ciphertext.

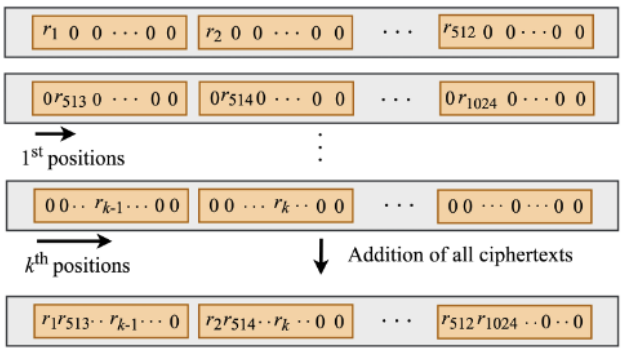

Figure 5: Compression of the authentication results.

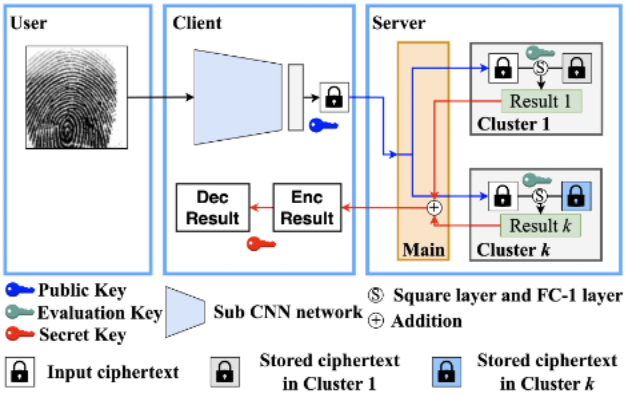

Figure 6: Cluster architecture for Blind-Touch.

The one-hot vector sets the first element of each registered user's feature vector to 1 and fills the remaining elements with zeros (see Figure 5). The next ciphertext is rotated by one and added to the resulting ciphertext to create a single ciphertext containing the summation result. By repeating this process with subsequent ciphertexts, we can simultaneously transmit up to 8,192 authenticated results to the client using just one ciphertext. The overall compression process is illustrated in Figure 5.

Upon receiving the ciphertext containing the authentication result, the client decrypts and applies the *sigmoid* function to each element of the decrypted vector. The client then looks for the element's index that exceeds the threshold. The corresponding index divided by 16 yields the quotient (q) and the remainder (r). The index of the original ciphertext stored on the server can be computed as $512 \cdot r + q$. This process only requires one additional multiplication and a rotation while significantly reducing the number of ciphertexts transmitted to the client and the authentication time.

Cluster Architecture

The compression method proposed for Blind-Touch facilitates the concurrent processing of up to 512 fingerprint authentications. Consequently, the overall authentication time increases with $\lceil N/512 \rceil$, where N is the number of registered users. If N is considerably large, this can lead to potentially slow authentication times.

One viable strategy to expedite the authentication time in Blind-Touch is to distribute and store the fingerprints across multiple servers and then process them independently and in parallel. We refer to this strategy as the "*cluster architecture*." The proposed cluster architecture is illustrated in Figure 6. The server infrastructure consists of two distinct components: the main server and the cluster servers. Upon receiving the encrypted feature vector from the client, the main server relays it to the appropriate cluster servers. Within this structure, the registered (and encrypted) feature vectors are sequentially indexed and stored on each cluster server.

For illustration, imagine a situation where 1,536 fingerprints are dispersed across three cluster servers. The initial 512 fingerprints would be stored in Cluster 1, the subsequent set, ranging from the 513th to the 1024th fingerprints, would reside in Cluster 2, and the final set would reside in Cluster 3.

depth	1	2	3	4	5
Size (KB)	459	658	855	1,075	1,280

Table 1: Ciphertext sizes according to depth with d=16,384 and a log scale factor of 40 in the SEAL-Python library.

Each cluster undertakes encrypted fingerprint computations and executes a leftward rotation, ensuring no data overlap. Once computations are complete, the main server collects all resulting ciphertexts into a singular ciphertext and transmits it back to the client. Leveraging this parallel processing mechanism can substantially reduce the authentication time.

Implementation of Blind-Touch

The Siamese network in Blind-Touch consists of two sub-CNN networks that share the same structure and weights. Each CNN network extracts the feature vectors from fingerprint images. The input size of the CNN is 224×224. The CNN architecture is composed of five convolutional layers. Each layer has the same structure except for the size of the output channel. The ith convolutional layer has $32 \times i$ number of output channels with the same padding and the size of the stride is one. After the convolutional layer, a Batch-Normalization, Swish activation, and MaxPooling layer are applied. To prevent overfitting, we use 40% dropout during training. The subtracted two CNN architecture is used as an input layer for FC-16. The activation function for FC-16 is a square function, which enables a symmetricity of two CNN networks. Finally, FC-1 and the $sigmoid$ function are used to output the authentication decision.

We adopted the CKKS scheme (Cheon et al. 2017, 2018) because of its efficiency in arithmetic operations over floating-point numbers (Clet, Stan, and Zuber 2021). We used the SEAL-Python library (https://github.com/Huelse/SEAL-Python). In the CKKS scheme, *depth* is predetermined during key configuration. Ciphertext size increases with increasing *depth*. Table 1. shows the ciphertext sizes relative to *depth*. To improve Blind-Touch's efficacy, it is important to reduce the ciphertext size, which necessitates minimizing *depth*. We set the *depth* to only support three multiplications, which matches the number of multiplications needed for Blind-Touch.

Key and Ciphertext	Size
Public key	1.1MB
Galois key	117MB
Relinearization key	4.5MB
Secret key	559KB
Ciphertext	856KB

Table 2: Sizes of different keys and ciphertext.

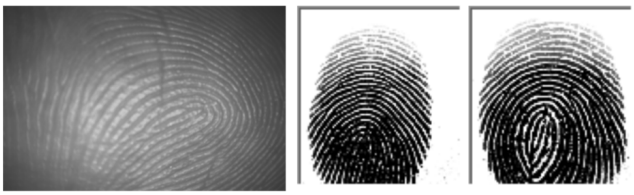

Figure 7: Samples from the PolyU dataset (on the left) and the SOKOTO dataset (on the right).

Given polynomial modulus degree d=16,384, the maximum number of encryptable elements is 8,192 (denoted as *number of slots*). Table 2 presents the sizes of the public key, Galois key, relinearization key, ciphertext, and the number of slots for the parameter settings d=16,384, the coefficient modulus is 240, and the log scale factor is 40 using the SEAL-Python library, which are computationally equivalent to 192-bit security in modern symmetric key encryption algorithms (Rahulamathavan 2022). These parameter configurations support up to three multiplications.

Evaluation

To evaluate the performance of Blind-Touch, we perform experiments on two publicly available popular benchmark fingerprint datasets: the PolyU Cross Sensor Fingerprint Database (Lin and Kumar 2018) and the SOKOTO Coventry fingerprint dataset (Shehu et al. 2018). Figure 7 shows some sample fingerprint images from each dataset.

PolyU Cross Sensor Fingerprint Database (**PolyU**): This dataset contains contact-based and contactless-2D fingerprint data. We use a processed version of the contactless dataset, which has two sessions. The first session includes 336 subjects, each with 6 fingerprint images, while the second session has 160 subjects, each with 6 images. From the 496 subjects, we choose 296 for training and the remaining 200 for testing, aligning with the dataset configuration in (Feng and Kumar 2023). The test set consists of 3,000 genuine and 19,900 imposter pairs.

SOKOTO Coventry Dataset (**SOKOTO**): This dataset comprises 6,000 fingerprint images, each measuring 96 $\times$ 103 pixels. We randomly partition this dataset into 3,600 fingerprints for training, 1,200 for validation, and 1,200 for testing. Each fingerprint image has a unique label, so we use one-shot learning (Koch et al. 2015). One-shot learning is a machine learning classification technique that involves training with only one example per class. Given the singular label in the SOKOTO dataset, we employ the pre-processing methods outlined in Table 3 to create genuine and imposter pairs. To form 1,200 genuine pairs, we apply each pre-processing technique to each fingerprint image. Subsequently, we select 8,400 imposter pairs, maintaining a 1:7 ratio between genuine and imposter datasets, which is similar to the PolyU dataset. Examples of the test samples are illustrated in Figure 8.

Figure 8: Test samples from the SOKOTO dataset: Original image (on the left) and its corresponding pre-processed versions (on the right).

Experimental Setup

We perform experimental evaluations using the metrics (Accuracy, F1-score, AUC score, and EER score) to understand the performance of Blind-Touch for real-world applications.

The overarching architecture of our system comprises three crucial components: 1 client, 1 main server, and 3 cluster servers. Five servers are used in service, each being a 'Standard-g2 Server' sourced from NAVER Cloud. Each server is powered by two cores (Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz) and equipped with 8GB of memory. The chosen servers perform similarly to standard personal computers, demonstrating the broader applicability of our study results. We incorporate the Python-based Flask (https://github.com/pallets/flask) framework to ensure seamless interactivity between these components.

We choose the hyperparameters (Epoch = 150, Adam optimizer) for optimizing the CNN model in Blind-Touch through experiments.

Authentication Accuracy

In Blind-Touch, the authentication result from the $sigmoid$ function ranges between 0 and 1. A threshold differentiates fingerprints from the same individual and those from different individuals. If the $sigmoid$ outcome exceeds this threshold, the fingerprint images are inferred to be from the same finger. The optimal threshold, dataset-specific and adaptively determined using a validation set, varies. The accuracy and F1-score of Blind-Touch at various thresholds are presented in Table 4 and 5.

Dataset	Train	Validation	Test
Dropout (%)	$0.01 \sim 0.15$	$0.01 \sim 0.15$	$0.01 \sim 0.15$
Scaling (%)	$90 \sim 110$	$90 \sim 110$	$90 \sim 110$
Translation (%)	$-10 \sim 10$	$-10 \sim 10$	$-10 \sim 10$
Rotation ($^\circ$)	$-30 \sim 30$	$-30 \sim 30$	$-30 \sim 30$
Gaussian blur	sigma = 0.7	sigma = 0.7	sigma = 0.7

Table 3: Pre-processing to generate test samples in the SOKOTO dataset with a dropout rate of 0.5 per channel.

Threshold	0.2	0.1	0.05	0.02	0.01	0.005
Accuracy	**98.3**	98.3	98.2	98.0	98.0	97.6
F1-score	**93.8**	93.6	93.3	92.6	92.0	91.3
AUC score	96.8	97.1	97.3	97.4	97.5	**97.5**
EER score	5.1	4.5	3.8	3.4	2.8	**2.5**

Table 4: Authentication accuracy of Blind-Touch on the PolyU dataset (presented as %).

Threshold	0.2	0.1	0.05	0.02	0.01	0.005
Accuracy	99.2	98.7	99.4	99.5	**99.5**	99.2
F1-score	97.0	97.7	97.8	98.0	**98.2**	97.0
AUC score	97.8	98.6	99.0	99.2	**99.4**	99.4
EER score	4.0	2.3	1.5	1.2	**0.7**	0.8

Table 5: Authentication accuracy of Blind-Touch on the SOKOTO dataset (presented as %).

The F1-score of Blind-Touch on the PolyU dataset peaks at 93.8% with a threshold of 0.2. On the SOKOTO dataset, the best F1-score of 98.2% is achieved with a threshold of 0.01. However, between thresholds of 0.2 and 0.005, Blind-Touch consistently maintains a high F1-score, exceeding 91% on the PolyU dataset and surpassing 97% on the SOKOTO dataset. These results suggest that Blind-Touch's authentication accuracy is not overly sensitive to threshold adjustments and guarantees high authentication accuracy even with a smaller feature vector size.

To demonstrate that Blind-Touch can maintain sufficient authentication accuracy when processing encrypted fingerprint data and remain competitive with state-of-the-art fingerprint authentication solutions designed for plaintext images, we compared the AUC and EER scores on the PolyU dataset against established solutions. Table 6 exhibits these comparison results. Blind-Touch achieved an AUC score of 97.5%, which is only 1.8% less than the leading results of ContactlessMinuNet (Zhang, Liu, and Liu 2021) and MinNet (Feng and Kumar 2023). For EER, Blind-Touch registered 2.5%, which is 0.6% higher than the other two methods. Our findings suggest that Blind-Touch serves as a viable alternative for applications sensitive to privacy.

Rank-1 Accuracy

To evaluate the feasibility of Blind-Touch for 1:N matching, we measured its Rank-1 accuracy on the distorted SOKOTO dataset, achieving 91%. This demonstrates Blind-Touch's potential for extension to 1:N matching tasks.

Execution Time and Storage Performance

We analyze the time and storage performance of Blind-Touch in comparison with the state-of-the-art solution, *DeepPrint* (Engelsma, Cao, and Jain 2019). We use cosine similarity for *DeepPrint*, following the parameters detailed in (Engelsma, Cao, and Jain 2019). Since *DeepPrint* does not provide code to generate feature vectors from fingerprints, we utilize the pre-generated 5,000 feature vectors

Method	AUC (%)	EER (%)
MNIST mindtct (Ko 2007)	58.9	36.9
MinutiaeNet (Nguyen, Cao, and Jain 2018)	92.0	13.4
VeriFinger (paid software)	98.2	3.0
ContactlessMinuNet (Zhang, Liu, and Liu 2021)	**99.3**	**1.9**
MinNet (Feng and Kumar 2023)	**99.3**	**1.9**
Blind-Touch (Ours)	97.5	2.5

Table 6: Comparison of the AUC and ERR scores for the PolyU dataset with the state-of-the-art fingerprint authentication solutions (in plaintext fingerprints).

Scheme	Input	Enc	Auth	Dec
DeepPrint w/ CKKS	62	2,635	772	4
DeepPrint w/ BFV	54	1,960	4,297	**1**
Blind-Touch (Ours)	**0.8**	**18**	**485**	8

Table 7: Blind-Touch vs. *DeepPrint* in input size (MB), encryption (Enc.), authentication (Auth.), and decryption (Dec.) times (ms).

available in their open-source project. To ensure a fair comparison, we conduct experiments using 5,000 fingerprints extracted from the PolyU dataset, allowing for duplicates.

Table 7 presents the comparative results between Blind-Touch and two versions of *DeepPrint* (CKKS and BFV). Experimental results show that Blind-Touch substantially outperforms *DeepPrint* in both execution time and storage performance, except for the decryption task. The input vector size for Blind-Touch is approximately 68 times smaller than that of *DeepPrint* w/ BFV. Moreover, encryption and authentication times are about 109 times and 9 times faster, respectively. Even when compared to the relatively faster authentication time of *DeepPrint* w/ CKKS, Blind-Touch is around 1.6 times faster. A drawback of Blind-Touch is its decryption time, which is roughly 8 times and 4 times slower than *DeepPrint* w/ BFV and *DeepPrint* w/ CKKS, respectively. However, an 8 ms decryption duration remains a negligible fraction of the total execution time. When summing up the times for all tasks in Blind-Touch, the total execution time of Blind-Touch is just 511 ms on average. However, the actual authentication duration is longer, extending to approximately 650 ms, when the feature extraction and network delivery times are considered. These results underscore that Blind-Touch is still well-suited for practical user authentication services, despite the integration of HE.

Ablation Study

An ablation study was conducted to evaluate the impact of key components in Blind-Touch.

FC-16 Evaluation after Encryption. The input size for FC-16 (25,088) is substantially larger than for FC-1 (16), resulting in computational overhead. To optimize the com-

putation overhead of these two FC layers, the first fully connected layer (FC-16) is computed in plaintext on the client, while the second fully connected layer (FC-1) is processed post-encryption on the server. This approach reduces the total evaluation time from 15,096 seconds to just 0.65 seconds.

Use of Compression Method. We evaluate the efficiency of our new compression technique. Consider N as the count of users (or feature vectors) stored on the server. When N surpasses 512, the server conducts $\lceil N/512 \rceil$ similarity checks, as each ciphertext holds a maximum of 512 feature vectors. Without this compression, separate ciphertexts would store each similarity check result, necessitating the server to send back $\lceil N/512 \rceil$ ciphertexts to the client for authentication. However, with our compression method, $\lceil N/512 \rceil$ ciphertexts can be merged into one if N is at most 8,192 since the authentication result occupies just one slot. For N greater than 8,192, ciphertexts reduce to $\lceil N/8,192 \rceil$ instead of $\lceil N/512 \rceil$. This technique keeps ciphertext size constant for up to 8,192 users or feature vectors. Implementing this method, we successfully reduced the ciphertext size from 2.6 MB to 0.26 MB for a 5,000 input test dataset.

Use of Clusters. The cluster architecture considerably impacts Blind-Touch's execution time. In Blind-Touch, a ciphertext can concurrently compare 512 fingerprints using SIMD operations. Hence, for 5,000 registered fingerprints, 10 similarity-matching comparisons, calculated as 10 = $\lceil 5,000/512 \rceil$, are required. This process is efficiently distributed across clusters for load-balancing. Table 8 illustrates the total authentication time for Blind-Touch, varying from 1 to 3 clusters, with 5,000 fingerprints registered.

Using a single cluster, Blind-Touch's authentication time is 1,334.4 ms, where one server handles all 10 comparisons. With two clusters, the load is split, with each handling 5 comparisons. In a three-cluster setup, two clusters manage 3 comparisons each, and the third handles 4, optimizing load distribution. Consequently, employing three clusters cuts the operation time by nearly half.

Optimization of Feature Vector Size. The size of the output feature vector impacts the computation time of the HE-based FC layer. The total time for Blind-Touch increased from 0.65 seconds with 16 features to 1.81 seconds with 64 features.

Optimization of CNN Model Architecture. To achieve a balance between performance and the overhead of HE, various neural network depths were tested on the PolyU dataset. The goal was to maintain high authentication accuracy while reducing model complexity. A 5-layer configuration is recommended, as it attained the highest F1 score of 93.8%, compared to 92.4% for the 4-layer and 89.3% for the 6-layer configurations.

# of clusters	1	2	3
Time	1,334.4	790.5	650.2

Table 8: Total authentication time (in ms) of Blind-Touch with different cluster counts for 5,000 registered fingerprints.

Security Analysis

We consider a curious server for the adversary model. The server can only see the ciphertext corresponding to the encrypted feature vector of a fingerprint received from the client and the ciphertext corresponding to the encrypted authentication result. In this section, we prove the security of Blind-Touch against a curious server using the simulation-based security in the semi-honest setting, which is widely employed to prove the security of protocols (Chandran et al. 2022).

We use $\mathcal{C}$ to represent a trusted client and $\mathcal{A}$ to represent an adversarial server. $\mathcal{A}$ wants to obtain information about the user's fingerprint data. For the proof, we generate a simulator $\mathcal{S}$ against the adversary $\mathcal{A}$ as follows.

The Simulator. When $\mathcal{C}$ encrypts the feature vector of a fingerprint image using the encryption key, the simulator generates encryption of 0s instead of the embedded feature vector. Note that the simulator has access to the public encryption key, which allows encrypting of any data.

Now, we define the following two games for the Blind-Touch framework π.

- The game $REAL_{(\pi,\mathcal{C},\mathcal{A})}$: The client $\mathcal{C}$ encrypts the feature vector extracted from a user's fingerprint and transfers it to the adversary $\mathcal{A}$.
- The game $IDEAL_{(\pi,\mathcal{S},\mathcal{A})}$: The simulator $\mathcal{S}$ encrypts a vector, which is entirely zero-populated, instead of the feature vector of the fingerprint and transfers it to the adversary $\mathcal{A}$.

We can prove the computational indistinguishability between $REAL$ and $IDEAL$ games as follows.

Theorem 2. *$REAL_{(\pi,\mathcal{C},\mathcal{A})}$ and $IDEAL_{(\pi,\mathcal{S},\mathcal{A})}$ are computationally indistinguishable.*

Proof. The two games differ only in one aspect is where, in $REAL_{(\pi,\mathcal{C},\mathcal{A})}$, $\mathcal{A}$ receives a ciphertext of feature vector of real user's fingerprint and, in $IDEAL_{(\pi,\mathcal{S},\mathcal{A})}$, $\mathcal{A}$ receives a ciphertext that is generated by encrypting the vector containing all zero elements. Regardless, $\mathcal{A}$ cannot distinguish two ciphertexts computationally because of the indistinguishability chosen plaintext attack (IND-CPA) security (Cheon, Hong, and Kim 2020) about HE. Both ciphertexts are computationally indistinguishable from the uniform random variable over the ciphertext space under the assumption of the hardness of the ring learning with errors (RLWE) problem (Lyubashevsky, Peikert, and Regev 2013). □

According to the claim, we can conclude that the difference in advantage between these two games is negligible. Therefore, $\mathcal{A}$ cannot obtain any information about the user's fingerprint. Consequently, Blind-Touch achieves security against the curious server.

Theorem 2 states that the difference in advantage between the two games is negligible. This means that $\mathcal{A}$ cannot obtain any significant information about the user's fingerprint. Therefore, Blind-Touch achieves security against the curious server.

Conclusion

As the prevalence of AI services grows, there are growing concerns about data privacy. Therefore, we must not only focus on improving the efficiency of machine learning models but also on ensuring that user data is processed securely and with privacy in mind. We introduce Blind-Touch, a distributed machine learning framework specifically designed for privacy-preserving fingerprint authentication using HE. Blind-Touch's optimized design enables efficient and accurate authentication, even in the presence of HE. Our findings provide essential guidance for the implementation of secure and privacy-preserving AI services.

Acknowledgements

The authors would thank anonymous reviewers. Hyoungshick Kim is the corresponding author. This work was supported by NAVER Cloud, the Korea Internet & Security Agency (KISA) grant (No. 1781000003, Development of a Personal Information Protection Framework for Identifying and Blocking Trackers) and the Institute for Information & communication Technology Planning & Evaluation (IITP) grants (No. 2022-0-01199, Graduate School of Convergence Security, and No. 2022-0-00688, AI Platform to Fully Adapt and Reflect Privacy-Policy Changes).

References

Acar, A.; Aksu, H.; Uluagac, A. S.; and Conti, M. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (CSUR)*, 51(4): 1–35.

Brakerski, Z. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, 868–886. Springer.

Brakerski, Z.; Gentry, C.; and Vaikuntanathan, V. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 1–36.

Chandran, N.; Gupta, D.; Obbattu, S. L. B.; and Shah, A. 2022. SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost. In *31st USENIX Security Symposium (USENIX Security 22)*, 1361–1378.

Cheon, J. H.; Han, K.; Kim, A.; Kim, M.; and Song, Y. 2018. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*, 360–384.

Cheon, J. H.; Hong, S.; and Kim, D. 2020. Remark on the security of ckks scheme in practice. *Cryptology ePrint Archive*.

Cheon, J. H.; Kim, A.; Kim, M.; and 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, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, 409–437.

Cho, G.; Huh, J. H.; Kim, S.; Cho, J.; Park, H.; Lee, Y.; Beznosov, K.; and Kim, H. 2020. On the security and usability implications of providing multiple authentication choices on smartphones: the more, the better? *ACM Transactions on Privacy and Security (TOPS)*, 1–32.

Chowdhury, A.; Kirchgasser, S.; Uhl, A.; and Ross, A. 2020. Can a CNN automatically learn the significance of minutiae points for fingerprint matching? In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 351–359.

Clet, P.-E.; Stan, O.; and Zuber, M. 2021. BFV, CKKS, TFHE: Which One is the Best for a Secure Neural Network Evaluation in the Cloud? In *Applied Cryptography and Network Security Workshops, Kamakura, Japan, June 21–24, 2021, Proceedings*, 279–300.

De Luca, A.; Hang, A.; Von Zezschwitz, E.; and Hussmann, H. 2015. I feel like I'm taking selfies all day! Towards understanding biometric authentication on smartphones. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 1411–1414.

Dowlin, N.; Gilad-Bachrach, R.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. 201–210.

Engelsma, J. J.; Cao, K.; and Jain, A. K. 2019. Learning a fixed-length fingerprint representation. *IEEE transactions on pattern analysis and machine intelligence*, 43(6): 1981–1997.

Fan, J.; and Vercauteren, F. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*.

Feng, Y.; and Kumar, A. 2023. Detecting locally, patching globally: An end-to-end framework for high speed and accurate detection of fingerprint minutiae. *IEEE Transactions on Information Forensics and Security*, 18: 1720–1733.

Folkerts, L.; Gouert, C.; and Tsoutsos, N. G. 2021. REDsec: Running Encrypted Discretized Neural Networks in Seconds. *Cryptology ePrint Archive*.

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 169–178.

Gootman, S. 2016. OPM hack: The most dangerous threat to the federal government today. *Journal of Applied Security Research*, 517–525.

Kim, T.; Oh, Y.; and Kim, H. 2020. Efficient privacy-preserving fingerprint-based authentication system using fully homomorphic encryption. *Security and Communication Networks*, 1–11.

Ko, K. 2007. User's guide to nist biometric image software (nbis).

Koch, G.; Zemel, R.; Salakhutdinov, R.; et al. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*.

Lin, C.; and Kumar, A. 2018. Matching contactless and contact-based conventional fingerprint images for biometrics identification. *IEEE Transactions on Image Processing*, 27(4): 2008–2021.

Lovisotto, G.; Turner, H.; Eberz, S.; and Martinovic, I. 2020. Seeing Red: PPG Biometrics Using Smartphone Cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Lyubashevsky, V.; Peikert, C.; and Regev, O. 2013. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 1–35.

Mare, S.; Baker, M.; and Gummeson, J. 2016. A study of authentication in daily life. In *Twelfth symposium on usable privacy and security (SOUPS 2016)*, 189–206.

Nguyen, D.-L.; Cao, K.; and Jain, A. K. 2018. Robust minutiae extractor: Integrating deep networks and fingerprint domain knowledge. In *2018 International Conference on Biometrics (ICB)*, 9–16. IEEE.

Rahulamathavan, Y. 2022. Privacy-preserving similarity calculation of speaker features using fully homomorphic encryption. *arXiv preprint arXiv:2202.07994*.

Rui, Z.; and Yan, Z. 2018. A survey on biometric authentication: Toward secure and privacy-preserving identification. *IEEE access*, 7: 5994–6009.

Shehu, Y. I.; Ruiz-Garcia, A.; Palade, V.; and James, A. 2018. Sokoto coventry fingerprint dataset. *arXiv preprint arXiv:1807.10609*.

Song, L.; Gong, D.; Li, Z.; Liu, C.; and Liu, W. 2019. Occlusion robust face recognition based on mask learning with pairwise differential siamese network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 773–782.

Wu, H.; Xu, Z.; Zhang, J.; Yan, W.; and Ma, X. 2017. Face recognition based on convolution siamese networks. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 1–5.

Yang, W.; Wang, S.; Yu, K.; Kang, J. J.; and Johnstone, M. N. 2020. Secure fingerprint authentication with homomorphic encryption. In *2020 Digital Image Computing: Techniques and Applications (DICTA)*, 1–6.

Zhang, Z.; Liu, S.; and Liu, M. 2021. A multi-task fully deep convolutional neural network for contactless fingerprint minutiae extraction. *Pattern Recognition*, 120: 108189.