HELPSE: Homomorphic Encryption-based Lightweight Password Strength Estimation in a Virtual Keyboard System

Michael Cho
University of Washington
Bothell, Washington, USA
mikec87@uw.edu

Keewoo Lee
Seoul National University
Seoul, South Korea
activecondor@snu.ac.kr

Sunwoong Kim
University of Washington
Bothell, Washington, USA
sunwoong@uw.edu

ABSTRACT

Recently, cyber-physical systems are actively using cloud servers to overcome the limitations of power and processing speed of edge devices. When passwords generated on a client device are evaluated on a server, the information is exposed not only on networks but also on the server-side. To solve this problem, we move the previous lightweight password strength estimation (LPSE) algorithm to a homomorphic encryption (HE) domain. Our proposed method adopts numerical methods to perform the operations of the LPSE algorithm, which is not provided in HE schemes. In addition, the LPSE algorithm is modified to increase the number of iterations of the numerical methods given depth constraints. Our proposed HE-based LPSE (HELPSE) method is implemented as a client-server model. As a client-side, a virtual keyboard system is implemented on an embedded development board with a camera sensor. A password is obtained from this system, encrypted, and sent over a network to a resource-rich server-side. The proposed HELPSE method is performed on the server. Using depths of about 20, our proposed method shows average error rates of less than 1% compared to the original LPSE algorithm. For a polynomial degree of 32K, the execution time on the server-side is about 5 seconds.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

cyber-physical systems security, homomorphic encryption, password strength estimation, virtual keyboard

ACM Reference Format:
Michael Cho, Keewoo Lee, and Sunwoong Kim. 2022. HELPSE: Homomorphic Encryption-based Lightweight Password Strength Estimation in a Virtual Keyboard System. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530338

GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA.

ACM ISBN 978-1-4503-9322-5/22/06.
https://doi.org/10.1145/3526241.3530338

1 INTRODUCTION

In recent cyber-physical systems (CPS), hardware devices are increasingly connected through networks [21]. In particular, a resource-rich cloud server is often used to overcome the speed and power problems of tiny edge devices. For example, the virtual reality (VR) system proposed by Kämäräinen *et al.* offloads the majority of rendering operations with high computational complexity to a cloud server [16]. This is helpful for acceleration if the data transfer bandwidth is kept low. However, the data is inevitably exposed to the cloud server and network. As such, security and privacy have become one of the main concerns in CPS design these days.

Cryptography is one of the critical solutions to CPS security issues. Specifically, if data is encrypted on a user's device before being sent to another device, it preserves privacy from the threat of attackers. For example, many recent wireless keyboards encrypt data to be transmitted using the popular Advanced Encryption Standard (AES) algorithm [8]. Although this is effective to preserve privacy in a network, decryption is required to perform operations on encrypted data. As a result, sensitive information is exposed to an untrusted device.

Homomorphic encryption (HE) enables one to perform operations on encrypted data, also called a ciphertext, without decryption [3]. Therefore, attackers on a cloud server cannot acquire any sensitive information from the ciphertext. Thanks to this feature, various HE-based real-world applications have been presented [13, 17]. In addition to HE, other cryptography techniques, such as secure multiparty computation, are also widely used in privacy-preserving computations [23, 24]. However, these techniques often require frequent and continuous data transfer between client- and server-sides, which is fatal in power-hungry CPS. HE has the advantage that no other transfer is required after encrypted data (and some keys) are initially sent. However, it has the disadvantage of being slow due to a large amount of computation. To solve this problem, various methods have recently been presented for acceleration, such as using GPUs [18] and custom hardware accelerators [19, 20], which makes large-scale end-to-end HE-based solutions working in real-time feasible.

In this paper, we propose a virtual keyboard (VKB) system networked with a cloud server for augmented reality (AR) and VR purposes. Specifically, a password entered through this system is encrypted using a HE scheme and then transmitted. The strength of this encrypted password is then securely estimated in the cloud server. As a password strength estimation algorithm, the lightweight password strength estimation (LPSE) algorithm proposed by Guo *et al.* is used [15]. In our proposed HE-based LPSE (HELPSE) method, the numerical methods proposed by Cheon *et al.* [5, 6] are adopted to support the division and comparison operation in

the LPSE algorithm. Our proposed VKB system with HELPSE is implemented as a client-server model using an embedded development board and evaluated in terms of accuracy, execution time, and memory footprint. In particular, the experimental results provide the number of iterations of the numerical methods that gives the accurate password strength estimation results for a practical HE parameter set.

2 BACKGROUND

2.1 LPSE Algorithm

The LPSE algorithm is based on the similarity between an entered password and a reference strong password [15]. A password is represented as a vector with the following five components in order: the numbers of digits, lowercase letters, uppercase letters, special characters, and the password length. Each component has a specific weight. In particular, weights for the five components are 1, 1, 2, 3, and 1 in order. For example, if the entered password is P!3b8u5$, the calculated password vector is (3, 2, 2, 6, 8). Each component value of the reference strong password vector is calculated to have equal probability. Suppose that we use a 94-key keyboard with 10 digits, 26 lowercase letters, 26 uppercase letters, and 32 special characters. If the length of a password is 18, the strong password vector is calculated as ($\lceil 18 \times \frac{10}{94} \rfloor \times 1$, $\lceil 18 \times \frac{26}{94} \rfloor \times 1$, $\lceil 18 \times \frac{26}{94} \rfloor \times 2$, $\lceil 18 \times \frac{32}{94} \rfloor \times 3$, 18) that is (2, 5, 10, 18, 18).

In this LPSE algorithm, a cosine-length similarity (CLS) is calculated to determine the similarity between an entered password and the reference strong password. Suppose that an entered password vector is $(x_1, x_2, x_3, x_4, x_5)$ and the reference strong password vector is $(y_1, y_2, y_3, y_4, y_5)$. The CLS value of the two vectors, denoted by $\mathrm{sim}(x, y)$, is then calculated as follows:

$$\mathrm{sim}(x, y) = \frac{\sum_{i=1}^{5}(x_i \cdot y_i)}{||X||\,||Y||} \cdot \frac{\min(||X||, ||Y||)}{\max(||X||, ||Y||)}, \tag{1}$$

where $||X|| = \sqrt{\sum_{i=1}^{5} x_i^2}$ and $||Y|| = \sqrt{\sum_{i=1}^{5} y_i^2}$. The CLS value ranges from 0 to 1. As the component values of the two vectors are similar to each other, the CLS value approaches 1.

Using the calculated CLS value, the password strength is estimated as follows: 1) *strong* if the CLS value is larger than or equal to 0.4; 2) *weak* if the CLS value is smaller than or equal to 0.19; 3) *medium* otherwise. The values of these thresholds are empirically chosen, and they vary depending on exceptional rules included in the LPSE algorithm. Note that the original LPSE algorithm uses an additional measurement called the password-distance similarity (PDS) along with CLS. However, PDS is not considered in this paper because the operations required for PDS calculation are not much different from those required for CLS calculation and both calculations can be done independently.

2.2 Numerical Methods in HE

Even though HE has the advantage of enabling operations on ciphertexts without decryption, several problems hinder practical real-world applications in the HE domain. First, HE (for word-wide) supports only addition and multiplication of ciphertexts. Subtraction of ciphertexts that is a variant of addition is also available.

Algorithm 1 $\mathtt{Inv}(x; d_i)$ [6]

Input: $x \in (0, 2)$, $d_i \in \mathbb{N}$
Output: an approximate value of $1/x$
1: $a_0 \leftarrow 2 - x$
2: $b_0 \leftarrow 1 - x$
3: **for** $(i = 0; i < d_i; i = i + 1)$ **do**
4: $\quad b_{i+1} \leftarrow b_i^2$
5: $\quad a_{i+1} \leftarrow a_i \cdot (1 + b_{i+1})$
6: **end for**
7: **return** a_{d_i}

Algorithm 2 $\mathtt{Comp}(x, y; n_c, d_c)$ [5]

Input: $x, y \in [0, 1]$, $n_c, d_c \in \mathbb{N}$
Output: a value between 0 and 1 (1 if $x > y$; 0 if $x < y$)
1: $a \leftarrow x - y$
2: **for** $(i = 1; i \le d_c; i = i + 1)$ **do**
3: $\quad a \leftarrow f_{n_c}(a)$
4: **end for**
5: **return** $(a + 1)/2$

Second, the maximum number of operations performed on a ciphertext is limited. This is because each operation increases noise used to hide a plaintext message in a ciphertext, and the noise size higher than a certain level precludes proper decryption. In particular, multiplication between ciphertexts rapidly increases the noise level, so the number of consecutive multiplications is particularly defined as the (multiplicative circuit) *depth*. This depth determines whether a real-world application is feasible in the HE domain.

To support arithmetic/logical operations other than addition and multiplication, several numerical methods have been presented [5, 6]. Algorithm 1 shows a numerical method for finding a multiplicative inverse used in division [6, 14]. This algorithm is based on the following formula to calculate $1/x$.

$$1/x = 1/(1 - (1 - x)) \approx \Pi_{i=1}^{d_i}(1 + (1 - x)^{2^{i-1}}), \tag{2}$$

where d_i is the number of iterations. If the value of x is between 0 and 2, the approximate value gets closer to the original value as the d_i value increases. There are two multiplications in each loop (lines 4 and 5 in Algorithm 1), but the depth is accumulated through an operand with a larger depth. Therefore, the depth of Algorithm 1 in the HE domain, denoted as $\mathrm{depth}_{\mathrm{inv}}$, is $d_i + 1$.

Algorithm 2 shows the numerical method for comparison operation between two numbers [5]. Suppose that two numbers to be compared are x and y. Then, the comparison result is calculated using the following formula: $\frac{f(x-y)+1}{2}$. Here, f is a step function with a function value of 1 if the input value is greater than 0, and -1 if the input value is less than 0. Since this step function cannot be performed in the HE domain, it is approximated as follows:

$$f_{n_c}(a) = \sum_{j=0}^{n_c} \frac{1}{4^j} \cdot \binom{2j}{j} \cdot a(1 - a^2)^j \tag{3}$$

The depth of Algorithm 2 in the HE domain, denoted as $\mathrm{depth}_{\mathrm{comp}}$, is approximately $d_c \lceil \log_2 n_c \rceil$. Typically, to obtain practical level of

Algorithm 3 Sqrt($x; d_s$) [6]

Input: $x \in [0, 1]$, $d_s \in \mathbb{N}$
Output: an approximate value of $\sqrt{x}$

1: $a_0 \leftarrow x$
2: $b_0 \leftarrow x \quad 1$
3: **for** $(i = 0; i < d_s; i = i + 1)$ **do**
4: $\quad a_{i+1} \leftarrow a_i \cdot (1 \quad \frac{b_i}{2})$
5: $\quad b_{i+1} \leftarrow \frac{b_i^2 \cdot (b_i \quad 3)}{4}$
6: **end for**
7: **return** a_{d_s}

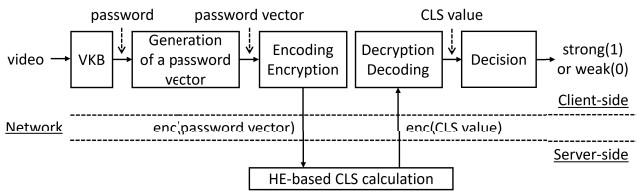


Figure 1: Overall architecture of a privacy-preserving VKB system with the proposed HELPSE method.

accuracy, Algorithm 2 that includes a nested loop requires a larger depth than Algorithm 1.

Algorithm 3 shows a numerical method to compute the square root of a real number [6]. A square root value is used to compute the min and max values between two numbers x and y as follows:

$$\min(x, y) = \frac{x + y}{2} \quad \frac{\sqrt{(x \quad y)^2}}{2}, \tag{4}$$

$$\max(x, y) = \frac{x + y}{2} + \frac{\sqrt{(x \quad y)^2}}{2}. \tag{5}$$

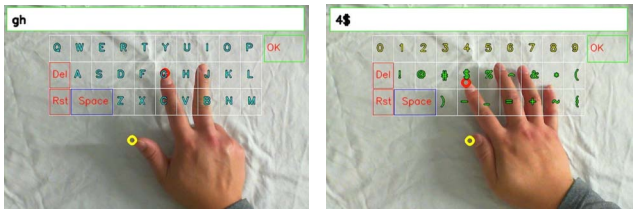
If the number of iterations of Algorithm 3 is d_s, the depth of this algorithm in the HE domain, denoted as $\text{depth}_{\text{sqrt}}$, is $2d_s \quad 1$. Since the square root operation of the min/max functions involves one square operation, their depth in the HE domain, denoted as $\text{depth}_{\text{min/max}}$, is $2d_s$.

Currently, there are three popular HE schemes [3]: BGV/BFV scheme [1, 11], CKKS scheme [4], and DM/CGGI scheme [7, 10]. Among them, the CKKS scheme supports real number arithmetic. Therefore, this HE scheme is most suitable for the numerical methods using real number operands. The CKKS scheme is available in well-known open-source HE libraries, such as Microsoft SEAL [2].

3 PRIVACY-PRESERVING VKB SYSTEM

As CPS, this paper considers a VKB system where a device with a single camera sensor receives characters and sends them to a server-side for processing. Specifically, the characters form a password. To evaluate a password while protecting it from the network and server-side, this paper applies a HE scheme to the VKB system. Specifically, a HELPSE method is proposed. This is a toy example, and the proposed approach can be used for other real-world applications running on similar CPS.

Figure 1 shows an overall architecture of a VKB system with the proposed HELPSE method. First, a VKB algorithm receives a



(a) (b)

Figure 2: The modified VKB layout. (a) ready gesture on the second-level layer with uppercase letters (b) click gesture on the third-level layer with digits and special characters.

password through streaming video input from a single camera. As a VKB algorithm, the method proposed by Lee *et al.* is used [22]. In this previous work, a keyboard layout is printed on a screen (of an AR/VR device), and the key on which the index finger is placed is entered by touching the thumb and index finger. This method enables fast typing because the typing gesture is simple and a user feels the touch of two fingers.

Although our proposed design employs the Lee's VKB algorithm to receive passwords, we had to significantly alter the interface and user input to add functionality missing from the original VKB layout with only lowercase letter keys. Namely, we changed the VKB algorithm to start with a familiar QWERTY layout, and integrated three total layers. By holding up the index, middle, and ring/little fingers of the right hand in a traditional numeric gesture, the layout is interchangeable from layers with lowercase alphabet letters, uppercase alphabet letters, and digits/special characters. Typing gestures are carried over from the original VKB algorithm by contacting the thumb and index finger. Figure 2 shows our modified VKB layout. In Figure 2(a), the index and middle fingers are stretched out, so uppercase letters are displayed in the layout. The uppercase letter G pointed to by the index finger is not yet entered because the thumb and index finger are apart. In Figure 2(b), all fingers are stretched out. Therefore, the displayed layout includes digits and special characters. Since the thumb and index touch, the special letter $ pointed to by the index finger is entered.

An entered password is converted into a password vector by the LPSE algorithm. In this conversion, the following exceptional rules presented in [15] are taken into account:

- Common 2-letter and 3-letter combinations are each calculated as one letter.
- Common starting and ending letters are not counted.
- Characters with a specific order are calculated as one character (e.g., qwerty, abcde, and 12345).
- Repeated characters are calculated as one character (e.g., aaaa, eeee, and 111).
- Special characters used like letters are calculated as letters (e.g., re@dy and cl!ck).

The five elements of a password vector are encrypted into separate five ciphertexts by the CKKS scheme and then transmitted to a cloud server. The weights used in vector conversion are multiplied to the encrypted password vector on the server-side and are assumed to be unknown to the client-side. Our proposed HELPSE, which is

described in the next section, calculates the CLS value from the encrypted password vector on the server. Since this CLS value is still encrypted, the server-side cannot acquire any information on the password.

4 SHALLOW DEPTH CLS FOR ENCRYPTED PASSWORDS

This section converts the previous LPSE algorithm into a HE-based one. To avoid bootstrapping that allows unlimited computations on a ciphertext but requires extremely huge computational complexity, we aim to complete all computations within a limited depth.

If CLS calculation shown in (1) is performed on an encrypted password using the numerical inverse and min/max algorithms, which is straightforward, a large depth is required. The details are as follows: 1) $||X||$ in the HE domain is first computed. Since the reference strong password vector is a constant one, $||Y||$ in the HE domain is computed in advance. The depth required in this step is $1 + \text{depth}_{\text{sqrt}}$; 2) The numerator and denominator of the CLS formula, $\sum_{i=1}^{5}(x_i \cdot y_i) \cdot \min(||X||, ||Y||)$ and $||X||\ ||Y|| \cdot \max(||X||, ||Y||)$, in the HE domain are computed simultaneously. These two computations additionally require the same depth of $1+\text{depth}_{\text{min/max}}$; 3) The inverse of $||X||\ ||Y|| \cdot \max(||X||, ||Y||)$ in the HE domain is computed by Algorithm 1, and the $\text{depth}_{\text{inv}}$ is added; 4) The results of steps 2) and 3) are multiplied to each other, so the final depth is $3+\text{depth}_{\text{sqrt}}+\text{depth}_{\text{min/max}}+\text{depth}_{\text{inv}}$. As described in Section 2.2, $\text{depth}_{\text{inv}}$, $\text{depth}_{\text{sqrt}}$, and $\text{depth}_{\text{min/max}}$ are $d_i + 1$, $2d_s - 1$, and $2d_s$, respectively. Therefore, the total depth is $3 + 4d_s + d_i$ if we use the same d_s value for $\text{depth}_{\text{sqrt}}$ and $\text{depth}_{\text{min/max}}$.

To reduce this total depth, our proposed method uses the numerical comparison algorithm instead of the numerical square root and min/max algorithms. It starts from the following equation:

$$\frac{1}{||X||\ ||Y||} \cdot \frac{\min(||X||, ||Y||)}{\max(||X||, ||Y||)} = \begin{cases} 1/||X||^2, & \text{if } ||X|| > ||Y|| \\ 1/||Y||^2, & \text{otherwise} \end{cases} \quad (6)$$

Suppose that $w = \texttt{Comp}(||X||, ||Y||; n_c, d_c)$. As the n_c and d_c values increase, the w value becomes close to 1 if $||X|| > ||Y||$, and to 0 if $||X|| < ||Y||$. Accordingly, the CLS formula is modified as follows:

$$\text{sim}(x, y) \approx \sum_{i=1}^{5}(x_i \cdot y_i) \cdot (w \cdot \frac{1}{||X||^2} + (1 - w) \cdot \frac{1}{||Y||^2}) \quad (7)$$

This equation does not require the min/max functions, but the square root function is still required to compute $||X||$ in the numerical comparison operation. However, squaring positive numbers does not change the comparison result, and therefore $||X||$ and $||Y||$ are replaced with $||X||^2$ and $||Y||^2$, respectively. Suppose that $w' = \texttt{Comp}(||X||^2, ||Y||^2; n_c, d_c)$. The CLS formula is then re-expressed as follows:

$$\text{sim}(x, y) \approx \sum_{i=1}^{5}(x_i \cdot y_i) \cdot (w' \cdot (\frac{1}{||X||^2} - \frac{1}{||Y||^2}) + \frac{1}{||Y||^2}) \quad (8)$$

The depth of the modified CLS formula in the HE domain is calculated as follows: 1) $||X||^2$ in the HE domain is computed, which requires a depth of 1; 2) $\texttt{Comp}(||X||^2, ||Y||^2; n_c, d_c)$ and $\texttt{Inv}(||X||^2; d_i)$ are computed simultaneously. The former usually consumes a larger depth than the latter, so the depth required in this step is $\text{depth}_{\text{comp}}$; 3) The remaining operations including two multiplications between ciphertexts are performed. Therefore, the total depth of the modified CLS formula is $3+\text{depth}_{\text{comp}}$ that is approximately $3+d_c\lceil\log_2 n_c\rceil$. If the n_c value is small, the modified formula provides more iterations, and consequently higher accuracy, under the same depth constraint compared to the original formula using the min/max and square root functions.

Table 1: Our HE Parameters for the 128-bit Security

Polynomial degree N	Bit-length of q	Maximum depth[†]
2^{15}	885	20

[†]40 bits are allocated to the majority of primes.

5 EVALUATION

This section evaluates our proposed VKB system with the HELPSE method. First, the experimental setup including HE parameters, test passwords, and development hardware is introduced. The depth, accuracy, execution time, and memory footprint of the proposed method are then presented.

5.1 Experimental Setup

The proposed HELPSE method is implemented using Microsoft SEAL open-source library [2]. In the CKKS scheme implemented in this library, two parameters mainly affect the implementation and performance of the target application: polynomial degree N and total bit-length of coefficients q. Table 1 shows the parameters of the CKKS scheme we used and the corresponding maximum depth. When a security level is fixed, N and (proper) $\log q$ are proportional [3]. In addition, as the size of q increases, the available depth increases. However, as the N value increases, the execution time increases significantly, and therefore the maximum N value in the current SEAL version is 2^{15}. The proposed HELPSE method uses this N value. For the 128-bit security that is popular for recent privacy-preserving real-world applications, the corresponding bit-length of q is 885 [2]. q is created as a product of distinct primes. As recommended by SEAL, if about 40 bits are allocated to (majority of) primes, 22 primes are generated. These primes are dropped by one whenever a multiplication between two ciphertexts is performed. Since the first and last primes are used for special purposes, the maximum depth of this parameter set is 20.

For evaluation, we collected 20 test passwords. Table 2 shows the collected passwords and their original CLS values compared to the reference strong password vector (2, 5, 10, 18, 18). The test passwords are categorized into four sets. Set 1 through Set 3 contain similar passwords. However, Set 1 includes very weak passwords, and Set 3 contains stronger passwords [12]. Set 4 includes very strong passwords, of which the average CLS value is 0.7 [9].

Our proposed VKB system with the HELPSE method, which is shown in Figure 1, is implemented as a client-server model. The client-side is implemented using a Raspberry Pi 4 Model B, including ARM Cortex-A72 and 8GB RAM, and a Microsoft Lifecam Cinema 720p Webcam. This implementation provides a good example of the use of low-cost and accessible hardware, and it can easily be adapted to a wide array of other devices and hardware. For the server-side,

Table 2: Test Passwords [9, 12] and Corresponding CLS Values by the Original LPSE [15]

Set 1		Set 2		Set 3		Set 4	
Password	CLS	Password	CLS	Password	CLS	Password	CLS
susan	0.135	Susan53	0.212	&Susan53	0.305	&Ru$p1gB0@Qx5F&ES1	0.759
jellyfish	0.254	jelly22fish	0.302	jelly22fi$h	0.309	PmC6!I1Bm@bsGiXR?q	0.717
ilovemypiano	0.349	!LoveMyPiano	0.400	!Lov3MyPiano	0.403	va@@WG@!agSOoy?pH6	0.676
Sterling	0.243	SterlingGmal2015	0.481	SterlingGmail20.15	0.606	I5R!BDgj5W3hpBpcQY	0.676
BankLogin	0.292	BankLogin13	0.344	BankLogin!3	0.347	wu0@lo#Ou3Dv5DKr7Z	0.671

Table 3: Total Depths and Average Error Rates ($d_i = 2$)

d_c	n_c	Total	Error rate (%)				
		depth	Set 1	Set 2	Set 3	Set 4	Average
2	2	10	12.37	11.62	11.04	3.21	9.56
2	3	12	7.45	7.52	7.26	2.82	6.26
2	4	14	4.16	4.59	4.54	2.45	3.94
3	2	13	3.21	3.69	3.70	2.31	3.23
3	3	16	0.35	0.60	0.73	1.39	0.77
3	4	19	0.02	0.06	0.13	0.70	0.23
4	2	16	0.11	0.23	0.35	1.06	0.44
4	3	20	0.00	0.00	0.01	0.16	0.04
5	2	19	0.00	0.00	0.01	0.15	0.04

a workstation with Intel Xeon W-2295 and 128GB RAM, working with Ubuntu 18.04 LTS, is used.

5.2 Depth and Accuracy

Table 3 shows the total depths and error rates of the proposed HELPSE method depending on the d_c and n_c values of Algorithm 2. In this table, the value of d_i of Algorithm 1 is fixed at 2 because values larger than 2 do not significantly affect the results. The first and second columns show the d_c and n_c values, and these values are set to meet the maximum depth under our HE parameter setting. The third column shows the total depths required in the proposed HELPSE method. The next columns show the error rates compared to the original non-HE-based LPSE algorithm [15]. Specifically, the last column shows the average error rates of the four password sets. The error rates are calculated as follows:

$$\text{error rate (\%)} = \frac{||\text{original CLS} - \text{approximate CLS}||}{\text{original CLS}} \times 100 \quad (9)$$

As the d_c and n_c values increase, the total depth increases because more consecutive multiplications between ciphertexts are performed. However, the measured depths do not match the ideal depths. This is because some SEAL functions we used for the implementation consume a depth even in situations where it is not necessary in order to provide simple and stable functions to beginners on HE. For example, multiplication between a ciphertext and a plaintext does not increase noise significantly, and thus depth consumption is usually not required. However, the corresponding SEAL function consumes a depth. For this reason, the total depths of the proposed method are slightly higher than the ideal depths.

Overall, the larger d_c and n_c values show the smaller error rates. Specifically, the d_c value has a greater effect on the error rates than the n_c value. For example, (d_c, n_c) of (3, 3) and (d_c, n_c) of (4, 2) require the same total depth of 16, but their average error rates are 0.77% and 0.44%, respectively. When the d_c value is 4, the error rates for all test sets are around 1% or less even if the values of n_c and d_i are 2. This implies that the proposed HELPSE method working with our practical HE parameters successfully adds password protection to the original LPSE method.

Table 4: Execution Time (Seconds) of LPSE Designs

Location	Client	Server			Client
Function	Enc.	Dec.	LPSE	Enc.	Dec.
Original LPSE [15]	-	-	2×10^{-5}	-	-
AES-based [15, 25]	0.02	0.004	2×10^{-5}	0.004	0.02
Proposed (3, 2, 2)	3.89	-	2.80	-	0.07
Proposed (4, 2, 2)	4.67	-	4.35	-	0.07
Proposed (5, 2, 2)	5.41	-	6.19	-	0.08

5.3 Execution Time

Table 4 compares the execution times of various LPSE designs. As previous designs, the original non-HE-based LPSE design and an AES-based LPSE design are used. To implement the AES-based design, an open-source code for 256-bit AES was used [25]. Similar to the proposed design, encryption and decryption are performed on the client-side, and LPSE is performed on the server-side. However, decryption on an encrypted password and encryption on the computed CLS value are additionally performed on the server to perform LPSE on a non-encrypted password, which exposes a password to the server-side. The proposed design in this table uses three (d_c, n_c, d_i) sets, (3, 2, 2), (4, 2, 2), and (5, 2, 2), of which depths are 13, 16, and 19, respectively. These depths affect the execution time, which is described in the last paragraph of this subsection.

Compared to the previous LPSE designs that are not based on HE, the proposed HELPSE design requires a longer execution time. However, this time is the cost of not exposing a password to the server-side. The original LPSE design is vulnerable on both network and server-side, and the AES-based design does not preserve a password on the server. The execution time of the proposed HELPSE design can be reduced by using other open-source HE libraries that support multi-threading or by hardware acceleration on the server. Although the transmission time in the network is not included in this table, the size of the transmitted ciphertexts is small (see the next subsection) and does not affect real-time processing.

As shown in the last three rows, an increase in depth in the proposed HELPSE causes an increase in execution time, which is also shown in the encoding/encryption functions running on the client-side. Specifically, if polynomial degree N is fixed, the execution time in HELPSE increases in proportion to the increase in depth. The time taken in the client functions can be shortened by using resource-rich hardware. For example, when executing the same client functions on the machine used for the server-side, the execution times for (d_c, n_c, d_i) of (3, 2, 2), (4, 2, 2), and (5, 2, 2) are 0.36, 0.43, and 0.50 seconds, respectively.

5.4 Memory Footprint

Typically, data with a large memory footprint in HE-based real-world applications are generated keys and input/intermediate/output ciphertexts. Generated keys are usually transmitted once, unlike input/output ciphertexts that should be transmitted whenever an input password changes, so they are not described in this subsection in detail. The size of intermediate ciphertexts may vary greatly depending on the implementation, and the memory allocated for them is freed frequently. Therefore, the memory footprint of input/output ciphertexts is only presented. To measure it, the encrypted password vector and CLS value were written as separate files. Under our HE parameter setting, the resulting CLS value that is contained in a single ciphertext was measured at 136 bytes. On the other hand, since a password vector has 5 components, the memory footprint of a password vector was measured at 680 bytes, which is usually not burdensome in many CPS.

6 CONCLUSION

This paper moves the previous LPSE algorithm to the HE domain to prevent password leakage in networks and a server-side. To perform operations in CLS calculations not supported by HE schemes, numerical methods are exploited. In particular, the existing CLS formula is modified to use the numerical comparison operation instead of the numerical min/max functions, which increases accuracy for a given depth. Our proposed HELPSE method is implemented as a client-server model. Specifically, the client-side is a VKB system for AR/VR, and a password is entered through this system. The previous keyboard layout is modified so that not only lowercase letters but also uppercase letters, digits, and special characters are entered. As a future work, we plan to extend our approach to other applications using passwords, where functions with more concealed information are executed on a server-side. In addition, we will apply the numerical methods for encrypted numbers to other CPS, such as a system using neural networks.

ACKNOWLEDGMENTS

We thank Dr. Rob Rutenbar, Dr. Jung Hee Cheon, Wonhee Cho, and the McNair Scholars Program for their continued support.

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00840, Development and Library Implementation of Fully Homomorphic Machine Learning Algorithms supporting Neural Network Learning over Encrypted Data). In addition, this material is based upon work supported by the National Science Foundation under Grant No. 2105373.

REFERENCES

[1] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*. Springer, 868–886.
[2] Hao Chen, Kyoohyung Han, Zhicong Huang, Amir Jalali, and Kim Laine. 2017. Simple encrypted arithmetic library v2. 3.0. *Microsoft Research, December* (2017).
[3] Jung Hee Cheon, Anamaria Costache, Radames Cruz Moreno, Wei Dai, Nicolas Gama, Mariya Georgieva, Shai Halevi, Miran Kim, Sunwoong Kim, Kim Laine, et al. 2021. Introduction to Homomorphic Encryption and Schemes. In *Protecting Privacy through Homomorphic Encryption*. Springer, 3–28.
[4] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 409–437.
[5] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient homomorphic comparison methods with optimal complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 221–256.
[6] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 415–445.
[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology* 33, 1 (2020), 34–91.
[8] Joan Daemen and Vincent Rijmen. 1999. AES proposal: Rijndael. (1999).
[9] Dashlane. 2021. *Resist hacks by using Dashlane's password generator tool.* Retrieved January 4, 2020 from https://www.dashlane.com/features/password-generator
[10] Léo Ducas and Daniele Micciancio. 2015. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 617–640.
[11] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.* 2012 (2012), 144.
[12] Paul Gil. 2021. *Examples of a Strong Password.* Retrieved January 4, 2020 from https://www.lifewire.com/strong-password-examples-2483118
[13] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.
[14] Robert E Goldschmidt. 1964. *Applications of division by convergence.* Ph. D. Dissertation. Massachusetts Institute of Technology.
[15] Yimin Guo and Zhenfeng Zhang. 2018. LPSE: lightweight password-strength estimation for password meters. *computers & security* 73 (2018), 507–518.
[16] Teemu Kämäräinen, Matti Siekkinen, Jukka Eerikäinen, and Antti Ylä-Jääski. 2018. CloudVR: Cloud accelerated interactive mobile virtual reality. In *Proceedings of the 26th ACM international conference on Multimedia*. 1181–1189.
[17] Sharmila Devi Kannivelu and Sunwoong Kim. 2021. A Homomorphic Encryption-based Adaptive Image Filter Using Division Over Encrypted Data. In *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 67–72.
[18] Sangpyo Kim, Wonkyung Jung, Jaiyoung Park, and Jung Ho Ahn. 2020. Accelerating number theoretic transformations for bootstrappable homomorphic encryption on GPUS. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 264–275.
[19] Sunwoong Kim, Keewoo Lee, Wonhee Cho, Jung Hee Cheon, and Rob A Rutenbar. 2019. FPGA-based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 1–8.
[20] Sunwoong Kim, Keewoo Lee, Wonhee Cho, Yujin Nam, Jung Hee Cheon, and Rob A Rutenbar. 2020. Hardware architecture of a number theoretic transform for a bootstrappable rns-based homomorphic encryption scheme. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 56–64.
[21] Edward A Lee, Sanjit A Seshia, et al. 2011. Introduction to embedded systems. *A Cyber-Physical Systems Approach* 499 (2011).
[22] Tae-Ho Lee, Sunwoong Kim, Taehyun Kim, Jin-Sung Kim, and Hyuk-Jae Lee. to be published. Virtual Keyboards with Real-time and Robust Deep Learning-based Gesture Recognition. *IEEE Transactions on Human-Machine Systems* (to be published).
[23] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 619–631.
[24] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 19–38.
[25] Danilo Treffiletti. 2014. *Aes256.* Retrieved February 10, 2022 from https://github.com/Urban82/Aes256