BlindFilter: Privacy-Preserving Spam Email Detection Using Homomorphic Encryption

Dongwon Lee*†, Myeonghwan Ahn†, Hyesun Kwak†, Jin B. Hong‡, Hyoungshick Kim*
* Sungkyunkwan University, Suwon, Republic of Korea, hyoung@skku.edu
† Seoul National University, Seoul, Republic of Korea, {dongwonlee95, lightb0x, hskwak}@snu.ac.kr
‡ University of Western Australia, Perth, Australia, jin.hong@uwa.edu.au

***Abstract*—Spam filtering services typically operate via cloud outsourcing, which exposes sensitive and private email content to the cloud server spam filter. Homomorphic encryption (HE) can address this issue by ensuring that user emails remain encrypted throughout all stages of the spam detection process on the cloud server. However, existing HE-based approaches are computationally infeasible due to the nature of HE operations. This paper proposes BlindFilter, a distributed, lightweight, HE-based spam email detection approach that consists of clients and servers collaborating to perform spam detection operations securely. BlindFilter employs WordPiece encoding and a modified Naive Bayes classifier, mitigating the need for multiplications and comparisons that would be prohibitive in terms of computation when applied with HE. Our experimental results demonstrate the efficacy of BlindFilter, with F1 scores exceeding 97% across two public email datasets. Furthermore, BlindFilter proves to be efficient as it can process an email in an average of 482.78 milliseconds. Our analysis also reveals that BlindFilter is robust against model extraction attacks, in which malicious users attempt to deduce the features of BlindFilter from query-response pairs.**

***Index Terms*—Spam detection, Homomorphic encryption**

I. Introduction

Content-based *spam filtering* is a popular technique that analyzes the words in an email to determine whether it is spam. Many content-based spam detection techniques have been developed based on machine learning models [1], and numerous cloud-based spam filtering services are now available. However, such techniques inevitably require users to allow a classifier typically hosted on a remote server outside their trust boundary to scan their emails. This raises concerns about privacy and censorship because the classifier may be able to access sensitive information in the emails, such as personal details or political opinions [2].

Out of many privacy-preserving techniques, homomorphic encryption (HE) is a promising approach, enabling computation over encrypted data without decryption. However, previous methods [3, 4] using HE on top of existing spam detection techniques have two major issues. Firstly, spam classification operations have a huge computational overhead that would be unacceptable in real-world services. Conventional content-based spam filtering methods require a large number of string comparison operations to check whether spam keywords are present in a given email. However, string comparisons are computationally expensive in HE schemes. Secondly, a sophisticated attacker can guess the spam keywords[1] used for content-based spam filtering methods by intentionally including or excluding specific keywords and monitoring the spam detection results. If an attacker successfully guesses the keywords used for a classifier, they can craft evasive spam emails without using those keywords. Therefore, an additional layer of protection is needed to mitigate such attack strategies.

This paper introduces BlindFilter, a distributed, lightweight, HE-based spam email detection system. This system consists of clients and servers that work together to carry out spam detection operations (including multiplications and additions) in an encrypted form. We have redesigned a Naive Bayes (NB) classifier [5], primarily focusing on minimizing the number of HE multiplications, which significantly contributes to computational overhead. We also use a substring encoding technique called WordPiece [6] instead of using complete words to decrease the number of string comparisons required. In addition, we suggest using two fine-tuning techniques to improve the model performance and mitigate spam keyword guessing attacks: (1) adding pseudo quantization noise (PQN) [7] to the spam keywords data and (2) applying Kurtosis regularization [8] to the model parameters. Both techniques are designed to maintain high model performance while making the classification results indistinguishable from the uniform distribution.

We evaluated the efficacy of BlindFilter on two publicly accessible email datasets: the TREC 2007 spam track dataset [9] and the Enron-Spam dataset [10]. For the TREC 2007 dataset, consisting of 50,199 spam emails and 25,220 non-spam (ham) emails, BlindFilter achieves an F1 score of 97.015% utilizing merely 20% of the dataset for training and validation. Likewise, for the Enron-Spam dataset consisting of 17,171 spam emails and 16,545 ham emails, BlindFilter achieves an F1 score of 97.889%. Contrasting with existing HE approaches [3, 4] necessitating extended processing times surpassing tens of seconds per email, BlindFilter shows the potential for practical deployment in spam filtering, requiring only 482.78 milliseconds (ms) per email on average. Moreover, our experimental findings demonstrate that incorporating two defense mechanisms–adding PQN and randomizing classification scores–can increase the noise range

[1]Spam keywords represent the words frequently appearing in spam emails. Those words can be used as indicators to identify spam emails.

2575-8462/23/$31.00 ©2023 IEEE
DOI 10.1109/SRDS60354.2023.00014

by approximately 7.787 and 1.914 times, significantly making model extraction attacks harder. These outcomes suggest that BlindFilter is a viable, privacy-preserving spam email filter capable of replacing conventional content-based spam filters to protect user privacy without revealing email content. The contributions of this paper are as follows:

- Implementation of BlindFilter[2], a practical HE-based spam email detection method that incorporates WordPiece encoding and a modified Naive Bayes classifier;
- Demonstration of BlindFilter's outstanding performance, achieving F1 scores above 97% on two publicly available email datasets and exhibiting rapid processing times (482.78 ms on average per email). Our results highlight the superior performance of BlindFilter when compared to state-of-the-art models [4, 11];
- Introduction of two defense techniques, including PQN addition and classification score randomization, which increase the noise range and prevent model extraction attacks (such as guessing spam keywords).

II. Homomorphic Encryption (HE)

HE is an encryption scheme that enables third parties (*e.g.,* cloud service providers) to perform certain operations on encrypted data without decrypting them. Let m_1, m_2 be messages, Enc be the homomorphic encryption function, and f and f' be computationally feasible functions for two ciphertext inputs and plaintext inputs, respectively. Then it satisfies that $f(Enc(m_1), Enc(m_2)) = Enc(f'(m_1, m_2))$. There are two types of HE: fully homomorphic encryption (FHE) and partially homomorphic encryption (PHE). FHE, first proposed by Gentry [12], allows evaluating arbitrary circuits on encrypted data, whereas PHE allows only one type of operation. In this paper, we use an FHE scheme to construct a privacy-preserving spam filtering framework because an NB classifier requires addition and multiplication operations. An FHE scheme consists of four algorithms: key generation, encryption, decryption, and evaluation.

- **Key Generation:** Generate a tuple of the secret key, encryption key, and evaluation key.
- **Encryption:** A party encrypts its plaintext using the encryption key.
- **Decryption:** A party decrypts a ciphertext using the corresponding secret key.
- **Evaluation:** Given ciphertexts encrypted under the same encryption key, evaluate a circuit over the ciphertexts using the evaluation key.

Various FHE schemes exist, including TFHE [13], FV [14], and CKKS [15]. We selected the CKKS scheme for BlindFilter because it supports addition and multiplication over complex numbers. Conversely, TFHE and FV only support computations over binaries and integers. FV and CKKS both support the encryption of multiple plaintexts in a single ciphertext via the *packing* technique [16], allowing for Single Instruction/Multiple Data (SIMD) homomorphic computations. The plaintext modulus is denoted as p, and sets of integers and integers modulo p are designated as $\mathbb{Z}$ and $\mathbb{Z}_p$. The ring of integers of the $(2N)$-th cyclotomic field is defined as $R = \mathbb{Z}[X]/(X^N + 1)$ for a power of two N. Note, $\mathbb{Z}[X]$ is a polynomial ring over $\mathbb{Z}$, indicating elements of $\mathbb{Z}[X]$ are polynomials with integer coefficients. Therefore, the plaintext space of both the FV and CKKS schemes is $R_p = \mathbb{Z}_p[X]/(X^N + 1)$, the residue ring of R modulo p. Multiple data can be encoded in this plaintext, with the number of slots indicating the maximum data amount that can be packed in a single plaintext. In the FV and CKKS schemes, multiple messages are encoded before encryption and decoded after decryption for packing. In addition, a *scaling factor* is employed for data precision during the encoding process.

In all existing FHE implementations, multiplication operations are notably slower than addition operations. Specifically, in the CKKS scheme, addition proves to be more than five times faster than multiplication [17]. Furthermore, executing a large volume of multiplications necessitates a process known as *bootstrapping* due to the inherent limitation on the number of multiplications, or the *level*, within the HE scheme. Bootstrapping essentially rejuvenates the ciphertext noise by evaluating and then re-applying the decryption and re-encryption procedures within HE. During these procedures, it is necessary to evaluate linear transformations for encoding and decoding and modular reduction p function $f(x) = x \pmod p$ for a given plaintext modulus p. Since HE only supports addition and multiplication, it is essential to approximate it as a polynomial with a small degree [18, 19]. Due to the polynomial operation and linear transformations for encoding and decoding, bootstrapping incurs a high computational cost. In existing HE schemes, bootstrapping still takes tens to hundreds of seconds depending on the number of slots in the ciphertext and the number of multiplications [19, 20, 21]. Consequently, it remains the bottleneck operation when using FHE with a large number of multiplications.

III. Attack Models

We consider two types of attackers that are likely in practice: (1) the honest-but-curious (also known as semi-honest) email server [22, 23] providing a spam classifier and (2) the malicious spam email writer who wants to generate evasive spam emails.

Honest-but-Curious Email Server: The honest-but-curious email server aims to glean private and sensitive information from the email content during the spam filtering process, despite adhering to the correct email services. This model is practically feasible because maintaining a good reputation is crucial for service providers in order to retain customers [22, 23]. If a provider behaves in a harmful manner, customers are likely to switch to other providers. Moreover, the provider may face legal consequences as well. As such, an honest-but-curious email server faithfully carries out email services but covertly attempts to learn sensitive information from the email content during spam filtering. To counteract this potential

[2]https://github.com/loofahs-due-0e/BlindFilter

threat, we operate under the assumption that users securely keep their secret keys and the chosen HE scheme is secure.

Spam Email Writer: The goal of the spam email writer is to generate effective spam emails to evade detection. To do this, the spam email writer needs to obtain information about the features used in the classifier. Gao *et al.* [24] introduced the Substitution-Then-Comparison (STC) attack, which is a type of enumeration attack, that substitutes input values to learn features (*i.e.,* tokens) used in the model by analyzing the differences in the outputs where a "token" refers to an instance of a particular type of data such as a word, phrase, or sentence. Gao *et al.* also proposed a protocol-level solution to mitigate the STC attack, but their proposal would be unacceptable for HE applications because the protocol requires several comparison operations, which would make it impractical. To solve this problem, we propose two techniques that can hide the output feature values' distributions, making it difficult to perform enumeration attacks such as the STC attack. Another way the spam writers can exploit the model is to use fake features (which can be achieved through poisoning or hijacking attacks, or similar) instead of the original ones from the spam email to exploit the server running an HE-based spam filter. The server alone cannot check whether the email sender validly sends the encrypted features to the server because the email and its features are processed in an encrypted form using the HE scheme. Therefore, we also provide the verification procedure for the receiver to check whether the features extracted from the transmitted email are actually used for spam filtering on the server side.

IV. Overview of BlindFilter

We show the workflow of BlindFilter in Figure 1. During the model construction phase, the server builds an NB-based spam classification model using the token vectors representing words that appear in the spam and ham emails. NB classifiers are widely used as the most practical content-based spam filtering technique due to their simplicity and performance [25]. In the model construction phase, we obtain the weight of each token and a threshold value for an NB-based spam classifier. In the email transmission phase, we use two encryption schemes: (1) a generic encryption scheme to encrypt email content and (2) a fully homomorphic encryption (FHE) scheme to encrypt the token vector generated from the email. After receiving the encrypted email and token vector, the server runs a classification model with the encrypted token vector. Next, the server sends the classification result with the encrypted token vector and email to the receiver. The receiver decrypts them and verifies the validity of the token vector. This validation check is done by comparing the token vector received from the classification server against the one generated from the received email from the sender (from the email transmission phase). If they are the same, then the token vector is valid (*i.e.,* no tampering during the classification phase by the server); otherwise, an invalid token vector has been used for the classification, and the email is discarded.

A. Model Construction

The model construction for BlindFilter consists of optimizing the NB spam filter model, assigning weights to the token vector to compute conditional probability using the NB classifier, and setting a threshold to determine the classification result. These steps are optimized for the NB spam filter model in our implementation for demonstration, but other models can be used instead, which would follow similar steps (*i.e.,* for any chosen spam filtering model, we would require model optimization, weight assignments to tokens, and spam determination). The steps are further described below.

1) Modifying the NB Classifying Formula: NB spam filtering is to classify emails into two classes, spam, and ham, by using the words in an email. The key idea is that the probability of an email being spam is related to the occurrences of words in the email. Let $W = \{w_1, w_2, ..., w_n\}$ be the words contained in the email. $P(S)$ represents the probability of an email being spam; $P(H)$ represents the probability of an email being ham; $P(w_i|S)$ represents the probability that the word w_i appears in a spam email; $P(w_i|H)$ represents the probability that the word w_i appears in a ham email. Under the NB assumption[3], we can compute $P(S|W)$ as shown in Equation (1).

$$P(S|W) = \frac{P(S) \cdot \prod_{i=1}^{n} P(w_i|S)}{P(S) \cdot \prod_{i=1}^{n} P(w_i|S) + P(H) \cdot \prod_{i=1}^{n} P(w_i|H)} \quad (1)$$

We empirically compute $P(w_i|S)$ and $P(w_i|H)$ using known ham and spam emails in the training dataset, which serve as word weights in the NB classifier. Given an email, the classifier calculates $P(S|W)$ using Equation (1) and determines its spam status by comparing $P(S|W)$ with the predetermined threshold value t. Specifically, the email is classified as spam if $P(S|W) > t$, where t is determined to optimize the model performance during training.

Applying an FHE scheme to implement a conventional NB spam filter without any modifications is computationally infeasible. This is because a conventional NB spam filter requires $O(n)$ multiplications, as illustrated in Equation (1), which leads to a significant computational overhead under an FHE scheme, as discussed in Section II. To address this issue, we carefully modify the NB classifying formula to reduce the computational overhead by replacing multiplications with additions in the logarithmic domain. First, we take the inverse of the inequality: $1/P(S|W) < 1/t$. Since $1/P(S|W) = 1 + (P(H) \cdot \prod_{i=1}^{n} P(w_i|H))/(P(S) \cdot \prod_{i=1}^{n} P(w_i|S))$, the inequality formula can be reformulated by taking the logarithm on both sides, as shown in Equation (2).

$$\log P(H) + \sum_{i=1}^{n} \log P(w_i|H) - \log P(S) - \sum_{i=1}^{n} \log P(w_i|S) < \log\left(\frac{1}{t} - 1\right) \quad (2)$$

[3]Each word makes an independent and equal contribution to the outcome.

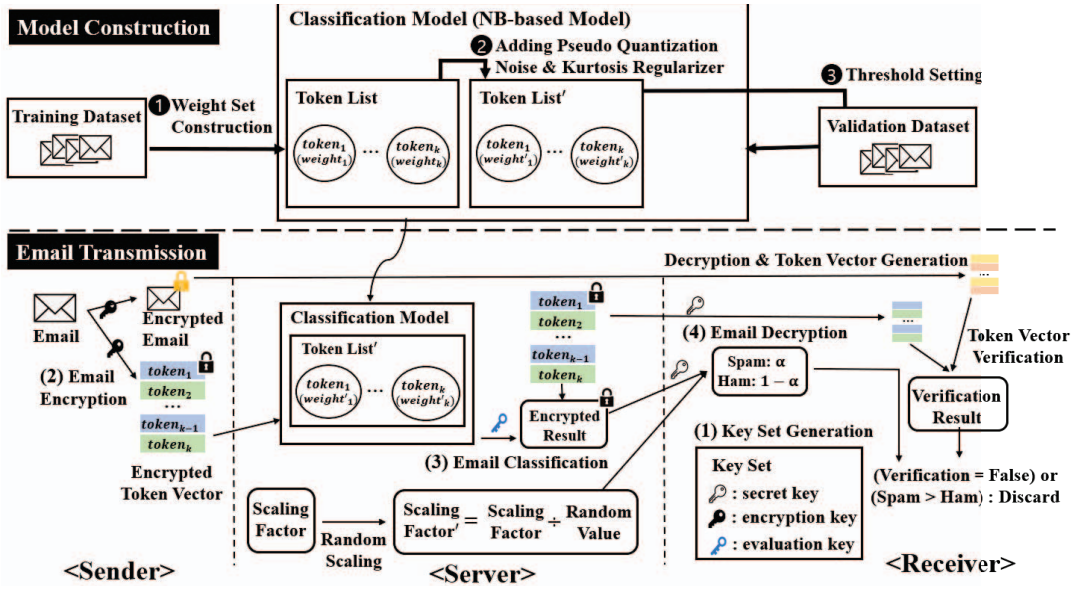


Fig. 1: BlindFilter workflow.

Experimental results show our modified NB approach–converting multiplications to additions as per Equation 2–is approximately 50 times faster than the original NB equation, which utilizes multiplications as shown in Equation 1, when processed under the HE implementation. As no approximation or omission occurs during the modification, it yields the same accuracy for spam filtering. The conversion overhead is negligible compared to the actual computation.

To construct the NB spam filtering model using this modified equation, the server's procedure consists of three phases, as shown in Figure 1: ❶ Weight Set Construction: We compute weight values of corresponding tokens, ❷ Adding PQN and Kurtosis Regularizer: We conceal the weight values of the tokens using PQN [7] and Kurtosis regularizer [8] to mitigate attack on the classification model, and ❸ Threshold Setting: determine a threshold which is the criterion to decide whether a new email is a spam or not.

❶ Weight Set Construction: To compute conditional probability using the NB classifier (*i.e.,* the left-hand side in Equation (2)), we need to know which features are included in the email content, implying that we have to compare two strings on the HE scheme when a message is encrypted itself. However, we cannot encrypt string-type data in an existing HE scheme. Khedr et al. [4] used a hash function to convert words into bit sequences before encryption, which requires a lot of memory to encrypt the entire message or features. Comparing two ciphertexts is computationally expensive under HE, even with few features. Moreover, a typical NB classification model uses whole words as the tokens for calculating $P(w_i|S)$ and $P(w_i|H)$, incurring significant computational overhead for processing tokens because we need to consider many words that might be contained in emails.

To address this problem, we encode words into substrings and use them as tokens instead of directly using words as tokens. This approach helps reduce the number of tokens used for an NB spam filter (and other content-based spam filters) and processing unseen words not observed in training because we no longer rely on the diverse and varying words from emails. Furthermore, we can control the number of tokens to use, significantly affecting the performance overhead of BlindFilter at the cost of classification performance. For BlindFilter, we use WordPiece [6], a widely used method in the natural language processing field. We use the set of 30,522 sub-word tokens in WordPiece embedding instead of the set of full words. This approach can achieve high performance (*e.g.,* accuracy) with a fixed number of slots. An example of WordPiece encoding is given in Figure 2, which shows the process of normalizing, and dividing words into substrings (*i.e.,* tokens) and their corresponding encoding values. We use normalization the same as BERT [26], including lowercasing characters and removing accents. The encoding is optimized for word comparisons using substrings, albeit not being human-friendly. We chose WordPiece as the encoding scheme for BlindFilter due to its popularity and feasibility of natural language processing.

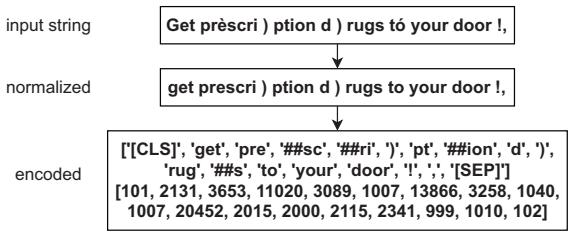


Fig. 2: Example of WordPiece encoding.

As shown in Equation (2), we must precompute $P(S)$, $P(H)$, $P(w_i|S)$, and $P(w_i|H)$ during the training phase. From the training set, we define $P(S)$ (respectively, $P(H)$) as the ratio of spam (respectively, ham) samples to the total number of samples. For $P(w_i|S)$ and $P(w_i|H)$, we calculate the frequency of appearance for each token w_i in the training spam and ham samples, respectively. For instance, if a token w_i appears in 15 spam emails among 100 spam emails, then $P(w_i|S) = 0.15$. In practice, the model stores the weight $\log P(w_i|H) - \log P(w_i|S)$ for each token w_i. We refer to the array of weights as the '*Weight Array* (WA).'

❷ **Adding PQN and Kurtosis Regularization:** To mitigate spam keyword guessing attacks such as the STC attack [24], we fine-tuned the weights with PQN [7] and Kurtosis regularization [8]. PQN is used for training and testing, while the Kurtosis regularization is used only for training. A PQN scheme was originally proposed for mixed-precision neural network quantization, where PQN enables quantization-aware training while finding optimal precision per unit of parameters. This scheme uses a pseudo quantized weight $\hat{w} = w + noise$ instead of the original weight w. Specifically, $\hat{w} = w + \frac{abs(w)}{(2^{num\_bits}-1)} * R$ where R is random variable, either uniform $U(-0.5, 0.5)$ or Gaussian normal $N(0,1)/2$. num_bits is trained per parameter in a differentiable manner. We use $abs(w)$ instead of $max(w) - min(w)$ because w is scalar. While previous work [7] used PQN to minimize the size of parameters for optimization, we use the same scheme, except for the actual quantization part, which is non-linear, to protect our classification model. Moreover, adding the PQN to NB weights helps confuse the outputs of the spam keyword guessing attack with the maximum available noise level per parameter value without compromising the performance. In our implementation, we use uniform noises rather than Gaussian noises because it produces better results.

Kurtosis regularizer [8] is used to make the distribution of tensors resemble the other, for example, uniform (Kurtosis value of 1.8), Gaussian (3.0), or Laplacian (6.0). We applied the Kurtosis regularizer on output activation. We intend to make output activation more like uniform distribution and thus be robust to the statistical attacks exploiting the distribution.

❸ **Threshold Setting:** The NB spam filtering process requires a threshold value of t, determining the output class. Since the threshold is directly related to the performance of NB spam filtering, it is crucial to establish a threshold value for classification. We select the threshold value that yields the highest F1 score through a simple linear search. The F1 score is the harmonic mean of precision and recall, where precision and recall are the ratios of true positives among positives and true positives among all correct classifications, respectively. We use a portion of the dataset as a validation dataset for evaluation. The server can construct the classification model with an email dataset, meaning there is no interaction with other parties. This implies that the construction can be executed in plaintext since there is no risk of feature leakage. Therefore, the server can build an NB classifier without heavy computational resources. Given the training and validation datasets, anyone can determine the optimal weights of the token list and threshold value t strictly offline. Moreover, there is no risk of revealing information about the model during training, validation, or noise addition. Consequently, these phases can be processed in plaintext rather than using computationally expensive HE methods.

B. Email Transmission

To ensure the confidentiality of the email, BlindFilter uses two encryption schemes: one is to encrypt the email itself, and the other is to encrypt the token vector for processing it with the NB-based spam classifier described in Section IV-A. Here, for simplicity, we focus on processing the encrypted token vector because email encryption is processed conventionally. As shown in Figure 1, the transmission phase of BlindFilter consists of four main steps as follows: **(1)** Key Set Generation, **(2)** Email Encryption, **(3)** Email Classification, and **(4)** Email Decryption. The key set generation is performed only once before the whole process (*i.e.,* precomputed). Details for each step are explained in the following.

(1) Key Set Generation: To utilize BlindFilter, the receiver generates a triplet consisting of a secret key, an encryption key, and an evaluation key. The recipient retains the secret key confidentially, whereas the encryption and evaluation keys are distributed to the sender and the server, respectively. In practical applications, establishing direct communication between sender and receiver for each email transmission to exchange the receiver's encryption key is challenging. As the encryption key is public data, it is more reasonable for the receiver to deposit it on the server beforehand (during the key generation stage). This enables any individual to access the receiver's encryption key for sending emails in a privacy-preserving fashion. Essentially, a person desiring to send an email to the receiver uses the corresponding encryption key. Importantly, given that the sizes of the secret and encryption keys are not considerable, the encryption keys for users who frequently participate in privacy-preserving email communications can be stored for future use.

(2) Email Encryption: A sender obtains the receiver's encryption key to encrypt a token vector. The sender then embeds the email content into tokens and creates an array named '*Email Array* (EA)', representing which tokens are included for more efficient computation on the HE scheme. The EA has slots filled with 1 for included tokens and 0 for the others. The EA is then encrypted with the receiver's public key and sent to the email server.

(3) Email Classification: The server receives the encrypted token vector from the sender. Still, it cannot infer any information about the email content without the secret key. The server processes the encrypted token vector with an evaluation key and classifies the email by computing two arrays, EA and WA. We note that the weights in WA are modified through fine-tuning with PQN and Kurtosis regularizer. For the classification decision, we compare the output value with the threshold

t. However, the comparison operations needed using an HE scheme would be computationally infeasible. Thus, the server subtracts the threshold from the output instead of using the comparison operation where the positive value indicates that the output is greater than the threshold. However, this output value can be a hint for a malicious client to guess the weights in WA. Therefore, we introduce a randomization technique to multiply the classification score with a random number from a distribution (*e.g.,* a normal or inverse Gaussian distribution) to avoid revealing raw classification scores. However, multiplying a random value requires HE multiplications, which increases the classification time. To prevent such a case, we propose a practical implementation technique using a randomized scaling factor instead, so that it has the same effect of multiplying a random value while reducing the depth of the classification circuit. Note that the scaling factor is already used in the CKKS scheme to ensure the precision of the message. It is multiplied in the encryption phase, and the evaluated ciphertext is divided by the scaling factor during the decryption phase. Therefore, we do not have to multiply a random value to the ciphertext in an HE manner, but instead, we can substitute it by dividing the scaling factor by a random value. Then, the decryption of the classification result shows the result multiplied by a random number.

(4) Email Decryption: The receiver can decrypt the encrypted classification result, token vector, and email with their own secret key. The decrypted classification result is used to determine if the email is spam; If the decrypted result is less than 0, it is classified as spam; otherwise, it is classified as ham. The receiver also checks that the decrypted token vector matches the one generated from the received email. Because of the indistinguishability of ciphertexts under an HE scheme, the server cannot detect whether the encrypted token vector is correctly generated from the original email. Still, the receiver can check this using the obtained email and the token vector by decrypting them. The final step is to ensure the validity of the token vector as follows.

1) The receiver decrypts the email and generates a token vector in the same way as the sender;
2) The receiver then compares this generated token vector with the token vector that was received and decrypted;
3) If the two token vectors match, the receiver accepts the email as valid;
4) If they do not match, the receiver discards the email. This is because a mismatch suggests that an incorrect token vector was used for spam filtering. This requires the sender to use a token vector that precisely represents the email's content.

V. Security of BlindFilter

As discussed in Section III, we consider two types of adversaries, namely a curious email server and a malicious spam email writer. The server can only see the ciphertext corresponding to the email content and the ciphertext corresponding to the classification results. In this section, we prove the security of BlindFilter against a curious server using the simulation-based security in the semi-honest setting, which is widely employed to prove the security of protocols (*e.g.,* [27, 28]). The security against the malicious spam email writer is evaluated experimentally in SectionVI-D.

Let $\mathcal{A}$ be an adversarial server that wants to obtain information about the user's email content. For the proof, we generate a simulator $\mathcal{S}$ against the adversary $\mathcal{A}$ as follows.

The Simulator. When the user encrypts email using the encryption key, the simulator generates encryption of 0s instead of the embedded email content. Note that the simulator has access to the public encryption key, which allows encrypting of any data.

Now, we define the following games for BlindFilter framework π with environment $\mathcal{Z}$.

- **The game** $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$: The BlindFilter framework π in the real world with environment $\mathcal{Z}$ and semi-malicious adversarial server $\mathcal{A}$.
- **The game** $IDEAL_{(\pi,\mathcal{S},\mathcal{Z})}$: It constructs the BlindFilter framework with the simulator $\mathcal{S}$. The difference from $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ is that it encrypts 0s instead of the embedded email content of the user.

We can prove the computational indistinguishability between real and ideal games as follows.

Claim. $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ and $IDEAL_{(\mathcal{F},\mathcal{S},\mathcal{Z})}$ are computationally indistinguishable.

Proof. The only difference between the two games is that the user encrypts the real input in $REAL_{(\pi,\mathcal{A},\mathcal{Z})}$ while it encrypts 0 in the game $IDEAL_{(\pi,\mathcal{S},\mathcal{Z})}$, if any. Regardless, the server cannot distinguish these two ciphertexts computationally because of the indistinguishability chosen plaintext attack (IND-CPA) security about HE. Both ciphertexts are computationally indistinguishable from the uniform random variable over the ciphertext space.

According to the claim, we can conclude that the difference in advantage between these two games is negligible. It implies that BlindFilter achieves security against the curious server.

VI. Experimental Results

In this section, we describe the analysis results of BlindFilter. Section VI-A explains the experimental environment, HE libraries, dataset, and HE parameters, which are used for performance evaluation. Then, the following sections show analysis results such as the performance (see Section VI-B), effects of parameter values on latency and throughput (see Section VI-C), and the effects of the proposed obfuscation techniques (see Section VI-D).

A. Experimental Setup

We implemented BlindFilter using the CKKS scheme over HE library 'Lattigo' (https://github.com/tuneinsight/lattigo) version 2.3.0 in Go language version 1.13.8. Both sender's and receiver's actions were conducted on the same machine (desktop with AMD Ryzen 7 3700X CPU and 64GB memory)

with an isolated system using a container, while the server's actions were conducted on a different machine using Intel 10400F and 32GB memory.

For experiments, we use the two publicly available email datasets: TREC 2007 spam track dataset (containing 50,199 spam and 25,220 ham emails) [9] and Enron-Spam dataset (containing 17,171 spam and 16,545 ham emails) [10]. As a default setting, we divide each dataset into training (10%), validation (10%), and testing (80%). We also set the number of slots to be 2^{15}. (*i.e.,* ring dimension of the ciphertext $N = 2^{16}$, which has twice the number of slots in the ciphertext.) To show changes in the performance and execution time of the BlindFilter under various conditions, we evaluate the performance for various training/validation/testing set splits and the ring dimension, respectively, in the following sections.

For the NB training, the training set is used to obtain weights of tokens to construct the weight set; the validation set is used to determine a threshold for classification as mentioned in Section IV. For the fine-tuning process, the training set is used to update parameters; the validation set is used to check the validity and measure statistics over training. The testing set is used only to measure the performance of BlindFilter.

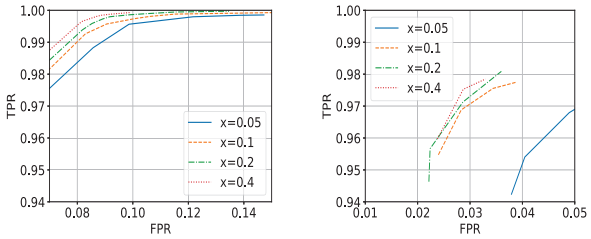
B. Performance

This section presents BlindFilter's performance in spam email classification, summarized in Table I, which includes the mean and standard deviation for each metric. BlindFilter achieves F1 scores over 0.97 for both datasets, even with encryption. On the other hand, the pre-fine-tuned NB model (*i.e.,* the baseline) performs adequately with the Enron-Spam dataset but poorly with the TREC 2007 spam track dataset. The baseline NB model's accuracy on TREC 2007 is 0.83071, demonstrating that fine-tuning significantly improves model performance, contributing to approximately a 13% increase in accuracy with a considerable noise margin equivalent to 0.17431 bit.

TABLE I: Performance of BlindFilter before and after fine-tuning.

Metrics	TREC 2007		Enron-Spam	
	Baseline	BlindFilter (std.)	Baseline	BlindFilter (std.)
Accuracy	0.83071	0.95965 (2.050e-3)	0.97868	0.97083 (2.227e-3)
F1 score	0.85990	0.97014 (3.441e-3)	0.97889	0.97117 (2.379e-3)
Precision	0.95815	0.95565 (1.150e-2)	0.98001	0.97031 (4.200e-3)
Recall	0.77992	0.98551 (1.693e-2)	0.97778	0.97210 (8.435e-3)
Bitwidth	-	0.17431	-	0.26405

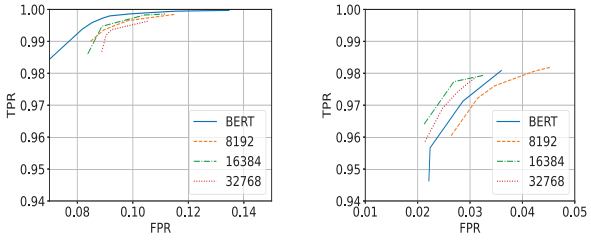
Effects of Training and Validation Dataset Sizes on the Performance: Figure 3 shows the true-positive rate (TPR) and false-positive rate (FPR) results of BlindFilter across various training/validation/testing set splits, where positive denotes spam. For comparative purposes, the tokenizer is fixed with the BERT tokenizer. Here, the fraction x means the dataset's proportion used for the training set ($x/2$) and the validation set ($x/2$), with the remaining $1-x$ used as the test set. As Figure 3 demonstrates, there are no substantial changes in TPR and FPR when $x \geq 0.2$. Hence, we use $x = 0.2$ in our subsequent experiments.



(a) TREC 2007 (b) Enron-Spam

Fig. 3: TPR and FPR results of BlindFilter with various training/validation/testing set splits.

Effects of the Number of Vocabulary Sizes Used in Tokenizer: Figure 4 shows the TPR and FPR results of BlindFilter with various vocabulary sizes for the tokenizer. Except for the BERT tokenizer, all tokenizers were trained to optimally encode the training set of each respective dataset. The custom-trained tokenizers underperformed the BERT tokenizer for all vocabulary sizes on the TREC 2007 dataset. For the Enron-Spam dataset, despite the custom-trained tokenizers with a size of 16,384 or greater outperforming the BERT tokenizer, due to the negligible performance gap and the BERT tokenizer's size (30,522) being less than 32,768, we default to using the BERT tokenizer unless otherwise specified.



(a) TREC 2007 (b) Enron-Spam

Fig. 4: TPR and FPR results of BlindFilter with various vocabulary sizes for the tokenizer. BERT represents a pre-trained BERT tokenizer, while the other labels represent the respective vocabulary sizes used for a custom-trained tokenizer.

Effects of the Modifying NB Equation: As presented in Equation (2) in Section IV, we modified the NB classification formula to enable efficient computation with HE, replacing multiplications with additions in the logarithmic domain. Table II shows that the modified formula used in BlindFilter significantly outperforms the original NB classification formula. In the model with 2^{14} slots, the computation time of the revised formula (104.98 ms) is approximately 52 times more efficient than that of the original formula (5,435.38 ms); in the model with 2^{16} slots, the computation time of the modified

formula (605.41 ms) is about 49 times more efficient than that of the original formula (29,478.22 ms). A minor modification (Equation (3)) to the original NB equation is necessary for HE computation by taking the inverse and multiplying the denominator term. This is because the original NB equation (Equation (1) in Section IV) includes divisions, which are not possible in HE.

$$P(H) \cdot \prod_{i=1}^{n} P(w_i|H) < (\frac{1}{t} - 1) \cdot P(S) \cdot \prod_{i=1}^{n} P(w_i|S) \quad (3)$$

TABLE II: Computation times of BlindFilter (in ms) where the NB equation is modified or not. The level of the ciphertext is 1 in the modified NB equation.

Model	Original (std.)	Modified (std.)
2^{14}	5,435.38 (75.33)	104.98 (1.56)
2^{15}	11,563.42 (141.86)	238.01 (3.64)
2^{16}	29,478.22 (374.78)	605.41 (5.13)

Effects of the Embedding Words: Unlike the state-of-the-art spam filtering approach using HE [4], which utilizes full words as tokens, BlindFilter utilizes WordPiece embedding [6]. As demonstrated in Table III, WordPiece embedding is significantly more effective than full words. With WordPiece embedding, BlindFilter achieves F1 scores of 0.97889 and 0.85990 for the Enron-Spam and TREC 2007 datasets, respectively, compared to 0.81661 and 0.65297 for models using full words as tokens. The F1 score of the model using full words does not improve when the word count exceeds $2,000$. These results highlight BlindFilter's superiority over the state-of-the-art spam filtering technique using HE [4] and the effectiveness of word embeddings, such as WordPiece embedding, in spam filtering.

TABLE III: Comparison of F1 scores for classification models with and without word embeddings.

Dataset	# of Words					WordPiece Embedding
	500	1000	2000	4000	8000	
Enron-Spam	0.74909	0.77758	0.81426	0.82991	0.81661	0.97889
TREC 2007	0.65283	0.65272	0.65360	0.65276	0.65297	0.85990

C. Execution Time

We evaluated the performance of BlindFilter using 500 randomly selected emails. We compared its execution time with a plaintext NB model to demonstrate its feasibility. For each email, we measured the time required for spam classification (computation time) and the time needed for transmitting the email with the corresponding classification result (transmission time). We examined three different HE implementations with varying the number of slots ($2^{14}, 2^{15}$, and 2^{16}) in the ciphertext. This analysis demonstrates how computation and transmission times change with the number of slots. Table IV shows the mean and standard deviation of computation and transmission times for BlindFilter.

TABLE IV: Transmission and computation times (in ms) for BlindFilter with varying the number of slots in the ciphertext. The times of the plaintext model are also provided.

Model	Transmission (std.)	Computation (std.)	Total (std.)
2^{14}	178.26 (51.20)	104.98 (1.56)	283.24 (51.34)
2^{15}	244.77 (24.94)	238.01 (3.64)	482.78 (25.20)
2^{16}	451.7 (25.23)	605.41 (5.13)	1057.11 (26.45)
plaintext	7.01 (2.25)	0.21 (0.039)	7.22 (2.25)

The transmission and computation times of BlindFilter increase linearly with the number of slots in the ciphertext, which corresponds to the ring dimension. A higher ring dimension results in a larger ciphertext size and, potentially, an increased ciphertext modulus associated with the computational complexity of individual operations. This means that the computation time of BlindFilter can vary significantly depending on the number of slots in the ciphertext. For example, when the number of slots is varied from 2^14 to 2^16, the computation time of BlindFilter ranges from 104.98 ms (with a standard deviation of 1.56 ms) to 605.41 ms (with a standard deviation of 5.13 ms). These findings indicate that BlindFilter is notably slower than the plaintext model, which requires about 7.22 ms per email. However, a processing time of 104.98–605.41 ms may still be acceptable for users prioritizing email privacy over quick delivery.

Table V presents the computation times of BlindFilter, considering variations in the number of slots in the ciphertext and the level of ciphertext. The maximum allowable number of multiplications is constrained by the ciphertext's level. The classification model's computation times for ciphertexts with levels 1, 2, and 3 are 238.01 ms, 408.13 ms, and 588.58 ms per email, respectively, when the number of slots is 2^{15} in the ciphertext. In summary, a classification model with a high ciphertext level and a large number of slots requires a longer computation time.

TABLE V: Computation times of BlindFilter (in ms) with varying the number of slots in the ciphertext and the level of ciphertext.

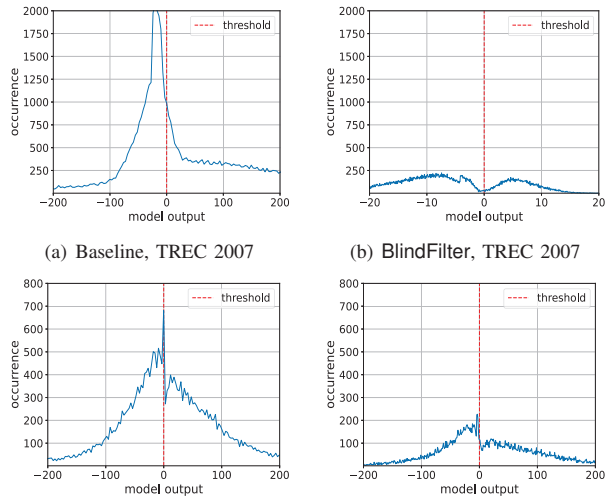
Model	Level 1 (std.)	Level 2 (std.)	Level 3 (std.)
2^{14}	104.98 (1.56)	185.45 (2.43)	285.86 (5.34)
2^{15}	238.01 (3.64)	408.13 (6.49)	588.58 (7.76)
2^{16}	605.41 (5.13)	942.56 (7.13)	1,392.44 (20.14)

D. Effects of PQN and Kurtosis Regularization

We analyze the impact of fine-tuning the weights with PQN [7] and Kurtosis regularization [8], as described in Section IV-A. The results of this analysis are shown in Figure 5. Figure 5(a) and 5(c) show the classification scores computed using the baseline model without fine-tuning (*i.e.,* PQN and Kurtosis regularization) on the TREC 2007 and Enron-Spam datasets, respectively. The classification score distribution resembles the Laplace distribution, potentially highly susceptible to feature inference attacks. Attackers could analyze output differences and infer token weights from the classification

score distribution, thereby learning the tokens used in the model.

BlindFilter implements fine-tuning of the model weights using PQN and Kurtosis regularization to make the classification outputs more challenging to analyze. As shown in Figure 5(b) and 5(d), adding PQN and Kurtosis regularization flattens the distribution of the classification scores, making it less predictable. The classification scores should ideally be evenly distributed to prevent attackers from exploiting the long tails of the distribution during STC attacks. The quantitative results of this analysis are provided in Table VI.



(a) Baseline, TREC 2007 (b) BlindFilter, TREC 2007

(c) Baseline, Enron-Spam (d) BlindFilter, Enron-Spam

Fig. 5: Distributions of classification scores for models with and without fine-tuning (PQN and Kurtosis regularization) on TREC 2007 and Enron-Spam datasets. Baseline: model without PQN and Kurtosis regularization; BlindFilter: fine-tuned model with PQN and Kurtosis regularization.

TABLE VI: Ablation study results on TREC 2007 and Enron-Spam datasets. Mean and standard deviation of results from 32 independent runs. D_{KL} represents the Kullback-Leibler (KL) divergence of classification score distribution from the uniform distribution ($U(-0.5, 0.5)$). The best results are highlighted in **Bold**.

Dataset	Metrics	Baseline	BlindFilter (std.)	Kurtosis only (std.)	PQN only (std.)
TREC 2007	Accuracy	0.83071	0.95965 (2.050e-3)	**0.96725** (1.875e-5)	0.96172 (4.628e-3)
	F1 score	0.85990	0.97014 (3.441e-3)	**0.97591** (1.450e-5)	0.97167(3.913e-3)
	Bitwidth	–	0.17431	8.0	**0.05528**
	Kurtosis	12.391	2.28803 (5.661e-3)	**2.20575** (6.854e-3)	2.70687(1.802e-2)
	D_{KL}	296.159	**0.97466** (4.201e-3)	0.99458 (3.301e-3)	1.28403 (1.500e-2)
Enron-Spam	Accuracy	0.97742	0.97095 (1.876e-3)	**0.97219** (1.589e-5)	0.96930 (2.572e-3)
	F1 score	0.97695	0.97139 (1.956e-3)	**0.97259** (1.609e-5)	0.96980 (2.706e-3)
	Bitwidth	–	0.60642	8.0	**0.22264**
	Kurtosis	21.200	2.99748 (4.592e-2)	**2.92963** (1.274e-4)	2.95973 (4.779e-2)
	D_{KL}	835.912	**1.35353** (5.515e-3)	1.37160 (5.716e-3)	1.37244 (6.556e-3)

We conducted an ablation study of the fine-tuning process to analyze the effectiveness of each component of BlindFilter. Our experimental results indicate that Kurtosis regularization alone produced the best accuracy and F1 score (with a low Kurtosis of 2.20575 on the TREC 2007 dataset and 2.92963 on the Enron-Spam dataset). However, this approach failed to achieve a low bitwidth, which is necessary to prevent spam keyword guessing attacks. On the other hand, using only PQN produced the lowest bitwidth, but the KL divergence increased compared with BlindFilter. Therefore, we recommend using both PQN and Kurtosis regularization (as implemented in BlindFilter) as the best-balanced solution regarding accuracy, F1 score, bitwidth, Kurtosis, and KL divergence.

Our experiments demonstrate that BlindFilter produced a mean bitwidth of 0.17431 for the TREC 2007 dataset and 0.60642 for the Enron-Spam dataset. This indicates that the noise range was around 7.787 and 1.914 times[4] larger than the original parameter for each dataset respectively. For instance, on the TREC 2007 dataset, if the original parameter value is 1.0, the expected noise range would be $[-3.893, 3.893)$, indicating that the parameter value used for computation would fall within the range of $[-2.893, 4.893)$. This suggests that more query-response pairs would be required for successful spam keyword-guessing attacks, indicating that the noise margin adds complexity to guessing the original parameter value from the encrypted data. Despite the substantial noise margin, BlindFilter still achieved an impressive F1 score of 0.97014 (with a low standard deviation of 0.00344) on the TREC 2007 dataset. Similarly, on the Enron-Spam dataset, BlindFilter achieved a high F1 score of 0.97139, with a low standard deviation of 0.00196.

We also examined the impact of the fine-tuning process on the distribution of classification scores. We calculated the Kurtosis of the classification score distribution and the KL divergence between the classification score distribution and the uniform distribution ($U(-0.5, 0.5)$) to quantify the distance between the two distributions. In the case of the TREC 2007 dataset, BlindFilter reduced the Kurtosis from 12.391 to 2.28803 and the KL divergence (D_{KL}) from 296.159 to 0.97466. Similarly, on the Enron-Spam dataset, BlindFilter reduced the Kurtosis from 21.200 to 2.99748 and the KL divergence (D_{KL}) from 835.912 to 1.35353, demonstrating that the fine-tuning process in BlindFilter effectively results in a distribution of classification scores that is more uniform, making it harder for attackers to infer the information about the tokens used in the model.

VII. Discussion

Effectiveness with a Small Training Dataset: As shown in Figure 3, varying training set sizes can impact model accuracy. Although performance improves as the training set size grows, no notable difference is observed when the combined proportion of the training and validation set size reaches 20%. Interestingly, BlindFilter achieves higher accuracy than the conventional word-based NB classifier even when trained with a minimal training dataset, indicating that the word embedding technique effectively constructs a token set for spam filtering. Moreover, unlike the traditional word-based NB, where the number of tokens may increase as new words

[4] $\frac{1}{2^{0.17431}-1} \approx 7.787$ and $\frac{1}{2^{0.60642}-1} \approx 1.914$.

are introduced over time, BlindFilter maintains a fixed size due to the utilization of a constant number of tokens for efficient HE computations.

Deployment Costs: Appendix A presents the key sizes for BlindFilter with respect to the number of slots and the ciphertext level, indicating that substantial storage and computational resources are required for managing the cryptographic keys. However, in BlindFilter, each client holds only the secret key, while the server is responsible for holding the evaluation and encryption keys. Consequently, we claim that BlindFilter can feasibly be deployed. A cloud server typically possesses greater computational power and storage capacity to manage the evaluation/encryption keys and perform operations with the evaluation key. Nevertheless, BlindFilter may introduce significant transmission overhead compared to traditional spam filtering techniques. In BlindFilter, the transmission time of the encryption key and encrypted data is approximately 35 times slower than that of plaintext data, as shown in Table IV. Thus, employing BlindFilter might not be advisable for environments requiring real-time processing. However, we surmise that latencies ranging from 300 to 3000 ms in most email applications would be acceptable for users.

VIII. Related Work

Privacy-preserving spam detection has been studied using various cryptographic approaches. Pathak *et al.* [29] used HE to address the classification problem via logistic regression. Feng *et al.* [30] devised a privacy-preserving Gated Recurrent Unit (GRU) network using a hybrid approach combining two cryptographic solutions: HE and garbled circuit. Wang *et al.* [31] constructed a deep learning model performing privacy-preserving classification on encrypted data utilizing functional encryption. However, these models share a common limitation: they necessitate continuous interaction between sender and receiver during computation for the scheme to function, which may only be feasible for specific applications.

Numerous proposals for an NB classifier with HE to safeguard users' privacy have been presented in prior studies [3, 4, 32, 33, 11]. While probability calculation is similar to the description in Section IV-A1, each proposal varies in its protocol. Wood *et al.* [32] necessitates interaction between a model owner and a data owner when identifying the maximum value of the result to assign the corresponding class. Yasumura *et al.* [33] assumes the presence of a trusted third party to handle such a task. In contrast, BlindFilter does not require interaction or a trusted third party during the training and spam filtering phases. Other research [3, 4] specifically concentrates on spam filtering applications, using spam keywords as features of the NB spam filtering model. To use such an approach, they need to perform string comparisons. Since no practical method exists for encrypting strings (*i.e.,* no HE library supports string-based operations), they utilize a hash function to convert words into bit sequences before performing comparisons (*i.e.,* they compute the OR (summation) of XOR (difference) on hashed values of the words to return 0 when two hashed values are identical, and 1 otherwise). However, this approach is impractical due to the computational exhaustion of comparing words individually. In Table III in Section VI-B, we demonstrate the efficacy of word embeddings compared to using full words as tokens, as in [3, 4]. By employing word embeddings, BlindFilter can efficiently handle token comparisons while maintaining the privacy and reducing the computational overhead associated with traditional string-based methods.

Recently, Nguyen *et al.* [11] introduced two spam filtering models based on HE and functional encryption (FE), respectively. Due to the unavailability of their open-source implementation, we cannot directly compare BlindFilter's results with theirs. However, we can indirectly compare the results since Nguyen *et al.* used the same TREC 2007 and Enron-Spam datasets. We allocated 20% of the dataset for training and validation, and 80% for testing, whereas they used 70% for training and 30% for testing. For the Enron-Spam dataset, Nguyen *et al.*'s model achieved an F1 score of 0.98, slightly higher than the 0.97 score achieved by BlindFilter, despite using a significantly larger training sample size. For the TREC 2007 dataset, they did not report F1 score results but presented accuracy metric results of approximately 0.98 when the feature vector size was 5,000, which is also slightly higher than the 0.97 achieved by BlindFilter. However, their HE (264.29 seconds) and FE (30.77 seconds) models demand considerably longer computation times to process an email compared to BlindFilter with 2^{15} slots, which requires only 0.24 seconds. This comparison highlights the efficiency advantage of BlindFilter in terms of computation time for processing emails while still maintaining a competitive performance in terms of F1 score and accuracy.

IX. Conclusion

Spam filtering is a crucial tool for email services; however, conventional content-based models inherently expose email content to the server, leading to privacy concerns. To address this issue, this paper presents BlindFilter, a privacy-preserving NB spam filtering model using FHE. BlindFilter is specifically designed to detect spam emails, even when encrypted, ensuring efficient and practical usage. Our experimental results demonstrate that BlindFilter achieves high F1 scores above 97% for two public email datasets, with an average execution time of 482.78 ms. These findings indicate that BlindFilter is a feasible privacy-preserving spam email filtering solution that can be implemented in real-world scenarios.

Acknowledgment

Hyoungshick Kim is the corresponding author of this research. The Institute of Computer Technology at Seoul National University provided the facilities for this work. The research was supported by several grants funded by the Korean government through the IITP: No. 2018-0-00532 (50%), No. 2022-0-00495, No. 2019-0-01343, and No. 2022-0-00688.

REFERENCES

[1] S. Kaddoura, G. Chandrasekaran, D. E. Popescu, and J. H. Duraisamy, "A systematic literature review on spam content detection and classification," *PeerJ Computer Science*, vol. 8, p. e830, 2022.

[2] A. AlMahmoud, E. Damiani, H. Otrok, and Y. Al-Hammadi, "Spamdoop: A privacy-preserving big data platform for collaborative spam detection," *IEEE Transactions on Big Data*, vol. 5, no. 3, pp. 293–304, 2019.

[3] L. J. M. Aslett, P. M. Esperança, and C. C. Holmes, "Encrypted statistical machine learning: new privacy preserving methods," *arXiv e-prints*, p. arXiv:1508.06845, Aug. 2015.

[4] A. Khedr, G. Gulak, and V. Vaikuntanathan, "Shield: Scalable homomorphic implementation of encrypted data-classifiers," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2848–2858, 2016.

[5] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering," *arXiv preprint cs/0006013*, 2000.

[6] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[7] A. Défossez, Y. Adi, and G. Synnaeve, "Differentiable model compression via pseudo quantization noise," *Transactions on Machine Learning Research*, 2022.

[8] M. Shkolnik, B. Chmiel, R. Banner, G. Shomron, Y. Nahshan, A. Bronstein, and U. Weiser, "Robust quantization: One model to rule them all," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[9] G. V. Cormack, "TREC 2007 spam track overview," in *Proceedings of The Sixteenth Text REtrieval Conference, TREC 2007, Gaithersburg, Maryland, USA, November 5-9, 2007*, ser. NIST Special Publication, E. M. Voorhees and L. P. Buckland, Eds., vol. 500-274. National Institute of Standards and Technology (NIST), 2007.

[10] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes - which naive bayes?" in *CEAS*, 2006.

[11] T. Nguyen, N. Karunanayake, S. Wang, S. Seneviratne, and P. Hu, "Privacy-preserving spam filtering using homomorphic and functional encryption," *Computer Communications*, vol. 197, pp. 230–241, 2023.

[12] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proc. of the Forty-First Annual ACM Symposium on Theory of Computing (STOC 2009)*, 2009.

[13] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *international conference on the theory and application of cryptology and information security*. Springer, 2016, pp. 3–33.

[14] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886.

[15] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Proc. of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2017)*, 2017.

[16] Z. Brakerski, C. Gentry, and S. Halevi, "Packed ciphertexts in lwe-based homomorphic encryption," in *Public-Key Cryptography – PKC 2013*, K. Kurosawa and G. Hanaoka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–13.

[17] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Lattigo: A multiparty homomorphic encryption library in Go," in *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, 2020, pp. 64–70.

[18] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2021, pp. 618–647.

[19] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *arXiv preprint arXiv:2106.07229*, 2021.

[20] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Advances in Cryptology – EUROCRYPT 2018*, J. B. Nielsen and V. Rijmen, Eds. Cham: Springer International Publishing, 2018, pp. 360–384.

[21] B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin, "When machine learning meets privacy: A survey and outlook," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.

[22] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *International Conference on Communications (ICC)*. IEEE, 2012, pp. 917–922.

[23] D. Liu, "Efficient processing of encrypted data in honest-but-curious clouds," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 970–974.

[24] C. zhi Gao, Q. Cheng, P. He, W. Susilo, and J. Li, "Privacy-preserving Naive Bayes classifiers secure against the substitution-then-comparison attack," *Information Sciences*, vol. 444, pp. 72–88, 2018.

[25] E. G. Dada, J. S. Bassi, H. Chiroma, A. O. Adetunmbi, O. E. Ajibuwa *et al.*, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[27] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography*, 2017.

[28] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.

[29] M. A. Pathak, M. Sharifi, and B. Raj, "Privacy preserving spam filtering," *arXiv preprint arXiv:1102.4021*, 2011.

[30] B. Feng, Q. Lou, L. Jiang, and G. C. Fox, "Cryptogru: Low latency privacy-preserving text analysis with gru," *arXiv preprint arXiv:2010.11796*, 2020.

[31] S. Wang, N. Karunanayake, T. Nguyen, and S. Seneviratne, "Privacy-preserving spam filtering using functional encryption," *arXiv preprint arXiv:2012.04163*, 2020.

[32] A. Wood, V. Shpilrain, K. Najarian, and D. Kahrobaei, "Private naive bayes classification of personal biomedical data: Application in cancer data analysis," *Computers in biology and medicine*, vol. 105, pp. 144–150, 2019.

[33] Y. Yasumura, Y. Ishimaki, and H. Yamana, "Secure Naïve Bayes classification protocol over encrypted data using fully homomorphic encryption," in *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*, 2019.

APPENDIX A
KEY SIZES FOR BLINDFILTER

Table VII shows the sizes of the encryption key, evaluation key, and secret key for BlindFilter with respect to the number of slots and the ciphertext level. This table provides insights into the storage and computational resources needed for managing and utilizing these keys in BlindFilter.

TABLE VII: Key sizes (in MB) for BlindFilter with the number of slots and the ciphertext level.

Model	Level	Encryption Key (MB)	Evaluation Key (MB)	Secret Key (MB)
2^{14}	1	2.1	31	1.1
	2	3.1	46	1.6
	3	4.1	61	2.1
2^{15}	1	4.1	65	2.1
	2	6.1	97	3.1
	3	8.1	129	4.1
2^{16}	1	8.1	137	4.1
	2	13	205	6.1
	3	17	237	8.1