Towards Practical Homomorphic Email Filtering: A Hardware-Accelerated Secure Naïve Bayesian Filter

Song Bian
Kyoto University
paper@easter.kuee.kyoto-u.ac.jp

Masayuki Hiromoto
Kyoto University
paper@easter.kuee.kyoto-u.ac.jp

Takashi Sato
Kyoto University
paper@easter.kuee.kyoto-u.ac.jp

ABSTRACT

A secure version of the naïve Bayesian filter (NBF) is proposed utilizing partially homomorphic encryption (PHE) scheme. SNBF can be implemented with only the additive homomorphism from the Paillier system, and we derive new techniques to reduce the computational cost of PHE-based SNBF. In the experiment, we implemented SNBF both in software and hardware. Compared to the best existing PHE scheme, we achieved 1,200x (resp., 398,840x) runtime reduction in the CPU (resp., ASIC) implementations, with additional 1,919x power reduction on the designated hardware multiplier. Our hardware implementation is able to classify an average-length email in 0.5 s, making it one of the most practical NBF schemes to date.

ACM Reference Format:
Song Bian, Masayuki Hiromoto, and Takashi Sato. 2019. Towards Practical Homomorphic Email Filtering: A Hardware-Accelerated Secure Naïve Bayesian Filter. In *24th Asia and South Pacific Design Automation Conference (ASPDAC '19), January 21–24, 2019, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3287624.3287699

1 INTRODUCTION

While outsourcing personal, organizational, or even institutional database to cloud services have become a common practice [7, 12], due to privacy concerns, the outsourced data are generally restricted to be in the less sensitive domains [7, 10]. As a perfect example, outsourcing personal or organizational email data to remote server presents to be challenging in terms of balancing privacy and efficiency. On one hand, since the server possesses a large amount training email samples, and thus a good email filter, the server should be able to filter ham/spam emails based on publicly-available word lists. On the other hand, in a traditional setup, this requires the server to have access to the email data, which violates privacy requirements in many situations [9].

As a result, the idea of implementing a secure email filter using advanced cryptographic primitives such as public-key encryption with keyword search (PEKS) [3] or partially or fully homomorphic encryption (PHE or FHE) [1, 4, 5, 13] has long been suggested. A secure email filter refers to the security scheme where, as Fig. 1 illustrates (ignoring the details), the server Chelsea is asked to classify an email from Bob to Alice as spam or not, without knowing the contents of the emails. The insecure version of this filter, where Chelsea knows the contents of the email, can be constructed using a naïve Bayesian filter (NBF) [15].

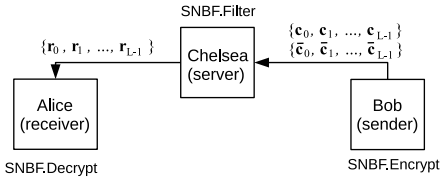

Figure 1: The proposed communication protocol.

The implementation of a secure email filter can thus be translated into a problem of realizing an NBF securely using advanced cryptographic primitives. While both PHE [2, 5] and FHE-based [13] approaches are suggested, the performance of general HE-based methods are still not in the practical domain. For instance, using the best-performing FHE schemes in [13] and [8], a multiplication between ciphertexts requires 300 ms and 17–22 ms, respectively, on a modern CPU. As shown in Section 5, this translates to more than 3 hours per word filtering in our setup, which is clearly impractical. By using interactive protocols and PHE, Bost et al. obtain improved efficiency, at the cost of high communication bandwidth [5]. In our experiment, we found that even using the existing PHE approach, we need around one minute to filter one word, whereas the length of emails in our test dataset averages to 287 words per email. Hence, improving the performance of existing approaches into a practical domain is the main focus of this work.

The goal of this study is thus to use hardware-assisted PHE-based techniques to construct a secure naïve Bayesian filter (SNBF) that is provably secure *and* practically deployable. The major contribution of this work is summarized as follows.

- **PHE-based SNBF**: We devised a novel secure filtering scheme where a client outsources PHE operations to a server, avoiding expensive FHE operations. Our technique works with all PHE schemes, including quantum-secure schemes based on the LWE problem (e.g., [11]). Specifically, we improve the existing naïve Bayesian filter [5] by deriving novel weight-embedding and batching techniques that reduce the total number of homomorphic operations by an order of 1,200x. The most important observation we make

ASPDAC '19, January 21–24, 2019, Tokyo, Japan

ACM ISBN 978-1-4503-6007-4/19/01…$15.00
https://doi.org/10.1145/3287624.3287699

is that, in the case of email filtering, we can simply embed rounded exponent weight into the matching result, instead of fixed-point representations used in [5, 13]. Hence, the proposed SNBF is a new approach for machine learning technique optimized for secure applications.

- **Accuracy and Practicality of SNBF**: To the best of our knowledge, we are the first to use combined hardware and software co-design to rigorously verify the accuracy and practicality of an SNBF for email classification. We verifies that the improved weight embedding does not impact the classification accuracy using real-world dataset, and with the assistance of application-specific hardware multiplier, we were able to securely classify an average-length email within 0.5 s, bringing secure email filtering into practical domain.
- **Architectural Design Exploration for Homomorphic Evaluation**: The major performance advantage in our SNBF comes from the hardware layer. By designing a specialized 2048-bit hardware multiplier based on the recursive Karatsuba-Montgomery algorithm, we were able to reduce the energy consumption of filtering a set of ciphertext (which may contain one or more words) scheme by 10^5 in magnitude compared to software implementation on CPU.

The rest of the work is organized as follows. First, in Section 2, we give short reviews on NBF and the Paillier cryptosystem. Next, the algorithmic-level and hardware-level descriptions of our proposed SNBF scheme are presented in Section 3 and 4, respectively. In Section 5, we discuss the accuracy and performance of SNBF. Finally, we conclude our paper in Section 6.

2 PRELIMINARIES

Throughout this work, we use the standard notation of $\mathbb{Z}_n$ to denote the residues of integer mod n. We use $\{x\}$ to refer to the set containing some number of elements, where the i-th element is x_i. Additionally, $\lg x$ is the shorthand for $\log_2 x$.

2.1 The Paillier Cryptosystem

In searching for a practically viable homomorphic encryption to be used in SNBF, we take a step back and look at the PHE described by Pascal Paillier [16]. Since we do not change the encryption and decryption scheme of Paillier, the reader is referred to the original paper [16] for the respective functions. We use Paillier.Enc for encryption function under Paillier, and Paillier.Dec for decryption function.

The important property of Paillier cryptosystem is its additive homomorphism, i.e., given the encryption of two messages $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$,

$$\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) \bmod n^2 = \mathsf{Enc}(m_1 + m_2 \bmod n). \tag{1}$$

Stemming from its additive homomorphism, the Paillier system permits the following operation on the ciphertext. For two messages m_1 and m_2,

$$\mathsf{Enc}(m_1)^{m_2} \bmod n^2 = \mathsf{Enc}(m_1 \cdot m_2 \bmod n) \tag{2}$$

$$\mathsf{Enc}(m_1) \cdot g^{m_2} \bmod n^2 = \mathsf{Enc}(m_1 + m_2 \bmod n). \tag{3}$$

In this work, we denote the addition in the ciphertext space (e.g., Eq. (3)) as $\boxplus$ (e.g., $\mathsf{Enc}(m_1) \boxplus m_2$), and (constant) multiplication as $\boxdot$ (e.g., Eq. (2) as $\mathsf{Enc}(m_1) \boxdot m_2$). The two properties will be extensively used to construct a secure keyword search scheme.

The Paillier cryptosystem is known to meet the indistinguishable under chosen plaintext attack (IND-CPA) security notion under the decisional composite residuosity assumption.

2.2 Naïve Bayesian Filter

The naïve Bayesian filter (NBF) is based on the simple application of Bayes's rule. First, in the training phase, we compute the probability $\Pr(s|w)$ for each word w as

$$\Pr(s|w) = \frac{\Pr(w|s)\Pr(s)}{\Pr(w)}, \tag{4}$$

where $\Pr(w|s)$ is the conditional probability of the appearance of the word w given that the email is a spam, $\Pr(s)$ is the probability that spam email occurs in all emails (*apriori*), and $\Pr(w)$ is the probability that the word occurs. The computed result $\Pr(s|w)$ is the probability of the email being a spam given that the word w appears in it. Note here that in an SNBF scheme, the training is done independently of the classifying phase, since it is assumed that the trained filter is public to the server.

To apply the model, assume that the incoming email contains the words $W = \{w_0, \ldots, w_{N-1}\}$, let p_{w_i} denotes the learned conditional probability $\Pr(s|w_i)$, the final probability that the email is spam can be computed as

$$\Pr(s_{\mathrm{MAP}}) = \frac{\prod_{i=0}^{N-1} p_{w_i}}{\prod_{i=0}^{N-1} p_{w_i} + \prod_{i=0}^{N-1}(1 - p_{w_i})}, \tag{5}$$

where the numerator is the probability that the email is a spam given the fact that all words in W simultaneously appear in the email as a set of independent random events, the denominator is the sum of the probability that the email is a spam given all the words in W, and $\Pr(s_{\mathrm{MAP}})$ refers to the maximum *a posteriori* probability for the email being a spam (note here that we assume $\Pr[s] = \Pr[h] = 0.5$ in the dataset. This value can be adjusted and made public by the server). Since the multiplication results in very small fractional numbers, a conventional way to calculate $\Pr(s_{\mathrm{MAP}})$ is to take the logarithm of both sides and rearrange the equation as

$$\eta = \sum_{i=0}^{N-1} (\log p_{w_i} - \log(1 - p_{w_i})) \quad \text{and} \tag{6}$$

$$\Pr(s_{\mathrm{MAP}}) = \frac{1}{1 + e^{-\eta}} \quad . \tag{7}$$

Equation (6) is essentially our target function to realize in a secure manner, and we use $\rho_i = \log p_{w_i} - \log(1 - p_{w_i})$ to simplify the notation. The user should be able to compute η after the secured classifications of the email.

3 SECURE EMAIL FILTERING: THE ALGORITHM

3.1 Notations

We use w for input word, and $\mathbf{w}$ for the binary-decomposed vector formed by bits in w. The bit-width of w, i.e., length of the vector

Algorithm 1 The SNBF.Filter function.

Require: $\mathbf{c} \in \mathbb{Z}_{n^2}^{\ell}, \{(v, \rho_v)\}$
1: **for each** $v_i \in \{(v, \rho_v)\}$ **do**
2: $c_{match_i} = 0$
3: **for each** $b_j \in \mathbf{v}_i, c_j \in \mathbf{c}$ and $\overline{c}_j \in \overline{\mathbf{c}}$ **do**
4: $c_{xnor} = ((c_j \boxdot b_j) \boxplus (\overline{c}_j \boxdot \overline{b}_j))$
5: $c_{match_i} = c_{match_i} \boxplus c_{xnor}$
6: $r_i = c_{match_i} \boxdot \rho_i$
7: Let $\mathbf{r} = [r_{N-1}, \ldots, r_1, r_0]$
8: **return** $\mathbf{r}$

Algorithm 2 The SNBF.Decrypt function.

Require: $\mathbf{r} \in \mathbb{Z}_{n^2}^{N}$
1: $\rho = 0$
2: **for each** $r_i \in \mathbf{r}$ **do**
3: $m_i = \text{Paillier.Dec}(r_i)$
4: **if** $\ell \mid m_i$ **then**
5: $\rho = m_i/\ell$
6: **return** ρ

$\mathbf{w}$, is denoted as ℓ. This bit-width also applies to the trained filter V, which is a collection of word-probability pair (v, ρ_v). Each v_i denotes a word, and $\rho_i = \log p_{v_i} - \log(1 - p_{v_i})$ is the Bayesian weight associated with v_i. For $\mathbf{w} = \{w_{\ell-1}, \ldots, w_0\}$, w_i means the i-th bit of w. We use $\mathbf{v}_i = \{b_{\ell-1}, \ldots, b_0\}$ to denote each bit of v_i. Thus, basically, we consider all elements in $\{v\}$ and w to have the same bit-width ℓ, which can be achieved by simple padding or hashing. The total number of word-probability pairs in V is N, i.e., $N = |V|$.

3.2 Communication Protocol and Algorithmic Construction

The basic setup of the SNBF is outlined in Fig. 1. Three parties are involved in one execution of the algorithm: the email sender Bob, the server Chelsea, and the email receiver Alice. As Fig. 1 depicts, Bob initiates the communication by using the function SNBF.Encrypt to encrypt the email as two encrypted vectors $\{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{L-1}\}$ and $\{\overline{\mathbf{c}}_0, \overline{\mathbf{c}}_1, \ldots, \overline{\mathbf{c}}_{L-1}\}$ (the need for $\overline{\mathbf{c}}$ will be later explained), where $(\mathbf{c}_i, \overline{\mathbf{c}}_i)$ represents the ciphertext of the i-th word in the mail with a total of L words to be transmitted. The ciphertexts are encrypted using Alice's public key, and sent to Chelsea for SNBF.Filter to classify. Each encrypted word $(\mathbf{c}_i, \overline{\mathbf{c}}_i)$ will be classified according to the filter, and the results, $\{\mathbf{r}_i\}$, are then sent to Alice, where SNBF.Decrypt is applied to decrypt each $\mathbf{r}_i$ to obtain ρ_{w_i}. Eq. (6) is then used to calculate $\Pr(s_{\text{MAP}})$. In what follows, the construction for each function is described. Note that our algorithmic construction computes on a per-word scale; hence, the repetitive application of the functions is required if multiple words are sent.

SNBF.Encrypt(w): takes as input a word w, and outputs the corresponding ciphertext $\mathbf{c}_w$. It simply breaks w into its binary representation $\mathbf{w} = \{w_{\ell-1}, \ldots, w_0\}$ where $w_i \in \mathbb{Z}_2$. w_i and its negated value $\overline{w}_i$ are encrypted using Paillier.Enc. The output will be two vectors $\mathbf{c}_w = \{c_{\ell-1}, \ldots, c_0\}$ where, as described, $c_i = \text{Paillier.Enc}(w_i)$, and $\overline{\mathbf{c}}_w = \{\overline{c}_{\ell-1}, \ldots, \overline{c}_0\}$ where $\overline{c}_i = \text{Paillier.Enc}(\overline{w}_i)$.

SNBF.Filter($\mathbf{c}_w, \overline{\mathbf{c}}_w, V$): takes as input two vectors of ciphertexts $\mathbf{c}$ and $\overline{\mathbf{c}}$, and the filter V. It outputs the probability-weighted result $\mathbf{r}_w$. The detailed algorithm is shown in Alg. 1. Upon receiving a pair of ciphertext vectors $\mathbf{c}_w$ and $\overline{\mathbf{c}}_w$, the algorithm starts to compute a bit-wise homomorphic match between the incoming ciphertext $(c_i, \overline{c}_i)$ and its own plaintext words $\{v\}$ in V. The filter function contains two loops. The outer loop compares each word $v_i \in \{v\}$ to the incoming word $(\mathbf{c}_w, \overline{\mathbf{c}}_w)$. The inner loop from Line 3 to 6 is evaluating a bitwise comparison between $(\mathbf{c}_w, \overline{\mathbf{c}}_w)$ and $\mathbf{v}_i$ (bit decomposed v_i). The algorithm then multiplies the Bayesian probability weight ρ_i into the matching result c_{match_i}, forming the i-th comparison result r_i. The results are concatenated into a vector $\mathbf{r}$, which is the output of the function. A subtle difficulty here is that ρ_i is a floating point number, and cannot be directly embedded into the ciphertext (homomorphic constant multiplication can only work with integers). Different from utilizing fixed-point representations in existing approaches, we thoroughly test the accuracy of NBF and discovered that by simply rounding the floating point number to the nearest integer, we can achieve almost no loss in classification accuracy (1% average difference). The demonstration for accuracy test will be conducted in Section 5.

SNBF.Decrypt: takes as input a ciphertext vector $\mathbf{r}$, output the probability weight ρ if input word w matches any of the server record, 0 otherwise. The algorithm for SNBF.Decrypt is sketched in Alg. 2. The loop in Alg. 2 takes each result in $\mathbf{r}$ and decrypts it using Paillier.Dec. The decrypted result m_i is then tested to see if it is a match, by dividing the bit length $\ell = |\mathbf{w}|$ into m_i. The correctness of this operation will be proved below, where we show that by dividing ℓ from the decrypted m_i, we can recover the integer probability weight ρ associated with the word w.

3.2.1 Correctness. A correctness proof for SNBF.Filter and SNBF.Decrypt is required to show that SNBF works as an email filter. First of all, SNBF.Encrypt is simple and the proof is left out because the function is simply several application of Paillier.Enc, which is correct as given. We use the following claim to construct a proof for SNBF.Filter and SNBF.Decrypt.

Claim 1. *Let ℓ be a prime, and $m_i = \rho_i \cdot k$ for some integer $k \in \{0, \cdots, \ell\}$. Assuming $\ell \nmid \rho_i$, if $\ell \mid m_i$, then $\rho_i = m_i/\ell$.*

The claim follows trivially Euclid's Lemma. However, an important note is that, as later shown in Section 5, ρ_i is a small number, and can be safely assumed to not be divisible ($\nmid$) by ℓ. The correctness for SNBF can then be proved through the following Lemma.

Lemma 3.1. *SNBF.Decrypt(SNBF.Filter(w)) $= \rho_w$ if $w \in V$, 0 otherwise.*

Proof. The important observation we make is that, since ciphertext-plaintext multiplication is available in Paillier, we can implement a perfect XNOR gate using PHE. Line 4 in Alg. 1 computes a single bit c_{xnor} by evaluating $((c_j \boxdot b_j) \boxplus (\overline{c}_j \boxdot \overline{b}_j))$, and the truth table is sketched in Table 1. Line 5 then adds the

Table 1: Truth Table for Homomorphic XNOR

c_j	$\overline{c}_j$	b_j	$\overline{b}_j$	$((c_j \boxdot b_j) \boxplus (\overline{c}_j \boxdot \overline{b}_j))$
Enc(0)	Enc(1)	0	1	Enc(1)
Enc(0)	Enc(1)	1	0	Enc(0)
Enc(1)	Enc(0)	0	1	Enc(0)
Enc(1)	Enc(0)	1	0	Enc(1)

results of bitwise XNOR together into one sum. Here, the result will be an encryption of ℓ if all bits result in matches, and $\{\text{Enc}(0), \text{Enc}(1), \cdots, \text{Enc}(\ell-1)\}$ if one or more bits result in mismatches. In Line 7 of Alg. 1, the weight ρ_i associated with v_i is embedded into the matching result. Thus, if w matches v_i, $r_i = \text{Enc}(\rho_i \cdot \ell)$. Otherwise, it is the encryption of the product of ρ_i and some value in the mismatched sums. By applying Claim 1, if r_i represents a match, which means that the decrypted $m_i = \rho_i \cdot \ell$, diving ℓ into the decrypted result, as in Line 3 in Alg. 2 recovers ρ_i. Otherwise, m_i will not be devisible by ℓ by Claim 1, giving an output of 0.

□

3.2.2 Security. In SNBF, what we want to protect is the content of the email, i.e., w from Bob to Alice. Hence, we want to make sure that the encrypted words are secure against Chelsea. Since PHE is used through the process of computation, it becomes relatively easy to see that the proposed scheme is IND-CPA against a honest-but-curious server (Chelsea).

3.3 The Batch Filtering Technique

For emails, it is generally the case that multiple words are concealed within. If SNBF.Filter is to be used as is, the main observation is that while a ciphertext can hold a huge integer ($\mathbb{Z}_n$ where n is at least 1024-bit integer), we are only using a tiny portion of it (the computed r_i is a product of ℓ and ρ_i, which will be on the scale of at most 10 bits). Consequently, we can pack more bits into one ciphertext for batch filtering, while not requiring any server-side change in computation.

To make the batch technique easier to understand, consider the following example. Let $\ell = 2$, for two words $\mathbf{w}0$ and $\mathbf{w}1$, we have $\mathbf{w}0 = \{w0_1, w0_0\}$ and $\mathbf{w}1 = \{w1_1, w1_0\}$. During SNBF.Encrypt, what we can do is to set $c_0 = \text{Enc}((w1_0 \ll \lceil \lg((\ell+1)\cdot\max(\{\rho\}))\rceil) + w0_0)$. Since $\ell = 2$, if we assume $\max(\{\rho\}) = 16$, $\lceil \lg((\ell+1)\cdot \max(\{\rho\}))\rceil = 6$. We can see that the contents of c_0, i.e., $\text{Dec}(c_0)$, can be described as

$$\text{Dec}(c_0) = 0\,0\,0\,0\,0\;w1_0\;0\,0\,0\,0\,0\;w0_0$$
$$\text{Dec}(\overline{c}_0) = 0\,0\,0\,0\,0\;\overline{w1_0}\;0\,0\,0\,0\,0\;\overline{w0_0}$$

Now consider we have a filter with only one word $\mathbf{v} = \{v_1, v_0\}$, where $v_1 = 1$ and $v_0 = 0$. According to Alg. 1, line 4 computes

$$c_{xnor0} = ((c_0 \boxdot 0) \boxplus (\overline{c}_0 \boxdot 1)). \tag{8}$$

In Paillier, we have $c_0 \boxdot 0 = c_0^0 = 1$, and $1 \boxplus (\overline{c}_0 \boxdot 1) = 1 \cdot \overline{c}_0^1 = \overline{c}_0$. Thus, $c_{xnor0} = \overline{c}_0$ for the zeroth bit. Similarly, we can see that $c_{xnor1} = c_1$ for the first bit. We then compute the match bit as $c_{match} = c_{xnor1} \boxplus c_{xnor0}$. If we look at the decrypted version of this homomorphic operation, we can see that it is basically computing the function

$$0\,0\,0\,0\,0\;\overline{w1_0}\;0\,0\,0\,0\,0\;\overline{w0_0}$$
$$+\;0\,0\,0\,0\,0\;w1_1\;0\,0\,0\,0\,0\;w0_1$$

Notice how both words $w0_0$ and $w1_0$ are simultaneously matched with v, even with different first bit. Now suppose $w1_1 = 1, w1_0 = 0$, and $w0_1 = 0, w0_0 = 0$. The above equation evaluates to

$$0\,0\,0\,0\,0\,\mathbf{1}\,0\,0\,0\,0\,0\,\mathbf{1}$$
$$+\;0\,0\,0\,0\,0\,\mathbf{1}\,0\,0\,0\,0\,0\,\mathbf{0}$$
$$=\;0\,0\,0\,0\,\mathbf{1}\,\mathbf{0}\,0\,0\,0\,0\,0\,\mathbf{1}$$

We assumed $\max(\{\rho\}) = 16$. Since we need $\ell \nmid \rho_v$, we assume $\rho_v = 15$ (The reason certain choice of ρ does not work in this simple example is because ℓ is too small. In general, $\ell > 32$, and there will be no $\rho \le 16$ where ℓ divides ρ), line 6 in Alg. 1 embeds the Bayesian weight into the result as

$$\mathbf{1\,1\,1\,1\,1\,0\,0\,0\,1\,1\,1\,1}$$

After decryption, we can examine the match/mismatch on a per-bit scale. We first take $01111 = 15$ out, and since $2 \nmid 15$, it is a mismatch. Then, for $11110 = 30$, $2 \mid 30 = 15$, which is a match and the corresponding weight $\rho_v = 15$.

While we do not have enough space to provide a proof, the batching technique is general and can be applied to as many words as a ciphertext can contain. In addition, all modifications required to deploy such a technique are all done to the encryption and decryption procedures SNBF.Encrypt and SNBF.Decrypt. Hence, the server does not even realize that multiple words are being processed in parallel. Using our instantiation parameters later described, we can embed around 100 words into one pair of ciphertext vector $\mathbf{c}$ and $\overline{\mathbf{c}}$, greatly improving the efficiency of the scheme.

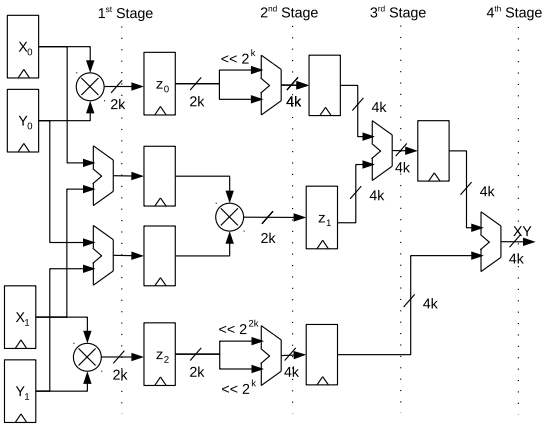

Figure 2: The architecture of a single recursive layer of the proposed RKM multiplier.

4 HARDWARE ARCHITECTURE

To implement a fast multiplier that meets the need of SNBF, we choose to use the recursive Karatsuba-Montgomery (RKM) multiplier based on the work in [6]. The major benefit of this approach is its flexibility and design simplicity. However, the work

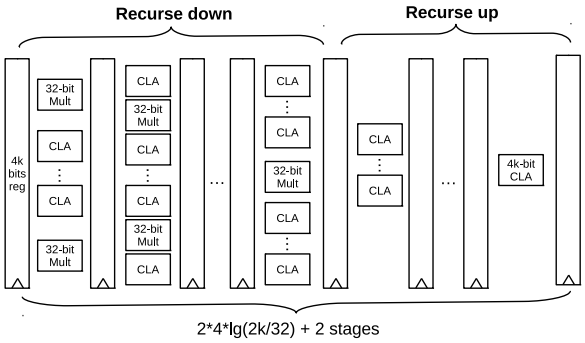

Figure 3: The general architecture of the proposed RKM multiplier with the recursive layers unfolded.

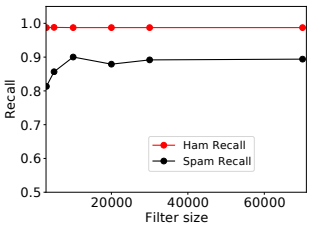

Figure 4: Ham/spam recall for different filter size N with floating point ρ.

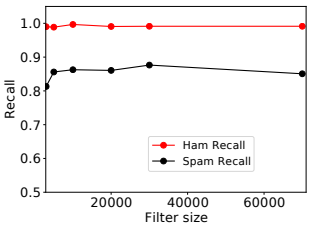

Figure 5: Ham/spam recall for different filter size N with integer ρ.

by Chow et al. uses an unoptimized version of the RKM algorithm, and only tested up to 512-bit multiplications due to the resource limitation on FPGA. To better evaluate the practicality of the proposed SNBF on a specialized hardware, we provide a full-custom design for the RKM multiplier.

In this work, we used an optimized version of the Karatsuba algorithm described as follows.

$$XY = (2^{2k} + 2^k)z_2 + 2^k(X_0 - X_1)(Y_1 - Y_0) + (2^k + 1)z_0,$$

where $z_2 = X_1Y_1$, $z_0 = X_0Y_0$. Figs. 2 and 3 outline the hardware architecture implementing a single recursive layer and overall architecture, respectively. For a single recursive layer, the algorithm only requires four pipeline stages. Unfolding the recursion down to 32-bit level, the number of required pipeline stages is $2 \cdot 4 \cdot \lg(2k/32) + 2$ in our example, since both the first and second stages will have $\lg(2k/32)$ levels of recursion for the multiplication. The third and fourth stages are single-stage operations, adding two to the total number of stages. For SNBF, the number of modular multiplications needed per filtering is quite large (over 70 K in our instantiation, as shown in Section 5). Therefore, even if we have a large number of pipeline stages, we can still get asymptotic single cycle performance. However, implementing the Montgomery reduction [14] algorithm requires three $2k$-bit multiplications. In this study, instead of instantiating three $2k$-bit recursive Karatsuba multipliers in parallel, we serialize the multiplications and use only one $2k$-bit multiplier. Hence, each $2k$-bit modular multiplication takes three cycles to complete.

5 EXPERIMENT

5.1 Experiment Setup

Throughout the experiment, we used the real-world Enron email dataset parsed by [15]. The email dataset contains six sets of emails that belong to six employees, pre-classified as ham or spam. The accuracy of the filter is trained using Enron1 to Enron5 (total of 27,716 emails), and tested on Enron6 (total of 6,000 emails). We also experiment on the best filter size N to minimize the amount of computation required. The accuracy of the SNBF is measured in ham and spam recalls, as in [15], which is traditional in measuring email classification accuracy. If the ham recall is low, it means a lot of ham emails are misclassified as spam emails, and similar analogy applies for spam recall. In general, users do not want low ham recall, and is more tolerant towards a lower spam recall.

For cryptographic instantiations, we used the Paillier scheme with 1024-bit key size, which is equivalent to an 80-bit security level. As a result, we need to instantiate our hardware to deal with 2048-bit integer multiplications, making $k = 1024$. Consequently, we have 50 pipeline stages in the design. We hashed the keywords into 61-bit bit strings, thus $\ell = 61$, which is a prime. 61 is chosen such that enough word space (number of unique words) is reserved.

The software version of SNBF is experimented using an Intel Xeon E5-2630 2.3 GHz processor on a single core with 32 GB memory. The 2048-bit RKM multiplier is synthesized on a commercial 65 nm low-power process node using a logic-synthesis tool [17].

5.2 Filter Accuracy

Figs. 4 and 5 are the results of the ham/spam recall when different filter sizes (N) are applied. The first important observation is that, larger N has diminishing effect on the classification recall, even if the filter is trained on different users' email sets. This property significantly reduces the amount of homomorphic computations required for SNBF while not hurting the accuracy performance. In evaluating the performance of SNBF, we chose to use $N = 10000$.

The second key result we found is that, while all existing studies related to NBF have been using fixed point representation for ρ [5, 13] in Eq. (6), as Figs. 4 and 5 compare, a rounded ρ basically has no significant impact on classification recall. While the maximum difference is by 3.8% in spam recall, which is itself insignificant (as discussed in [15]), the average difference is around 1% for spam recall, and well below 1% for ham recall. By losing one percent of accuracy on average, the efficiency is considerably improved. Instead of converting the weights into 52-bit integers as in [5], we can just round the exponents up and compute as is. In addition, since it is meaningless to have an extremely large exponent (e.g., the difference between $1/(1+e^{16})$ and $1/(1+e^{32})$ is negligible), we can set the maximum ρ to be less than or equal to 16 (the maximum ρ for all 10,000 filter words was found to be 17). By using this technique, the proposed filter greatly reduces the complexity to homomorphically combine the matching result and the weight ρ. We only need at most 7 large integer multiplication ($\mathsf{Enc}(\ell)^{3+2+1+0}$ for $\mathsf{Enc}(\ell) \boxdot 15$), instead of computing $\sum_{i=0}^{51}(\mathsf{Enc}(\ell) \boxdot 2^i)$ (translates to more than 1000 large integer multiplications) in [5], on a per word scale. A more concrete analysis is provided in the next section.

Table 2: Synthesis Result for RKM Multiplier

Bit Width	Power	Area (Gate Count)	Delay
2048	49.5 mW	18557560	69 ns

Table 3: Summary of the Performance Comparison SNBF and Other Existing Works

	[13]	[5]	CPU	RKM
Ctxt size	487.5 KB	512 B	512 B	512 B
Time/Mult	3.48 ms	9–29 μs	9–29 μs	207 ns
Time/Word	2.55 Ks	59.76 s	49.8 ms	0.15 ms
Power	165 W	95 W	95 W	49.5 mW
Energy/Word	420 KJ	5.67 KJ	4.73 J	7.42 μJ

5.3 Performance Comparison

As shown in Table 2, we were able to achieve 69 ns stage delay at 49.5 mW with area at around 18.5 M gates. Here, the stage delay is the longest path between two stage registers, which is optimized to be a single 32-bit multiplier. We achieve a throughput of 4.8 M 2048-bit multiplications per second.

To put the hardware performance data in context, we compare SNBF to the FHE-based scheme in [13] and PHE-based naïve Bayesian filter in [5] in Table 3. The SNBF result is the amortized cost by assuming that 100 words are concurrently processed using the batch technique mentioned in Section 3.

The total number of ciphertext multiplications (homomorphic addition) required for filtering one encrypted word through the 10,000-word NBF is 734,147. The majority of the multiplications comes from Line 5 in Alg. 1, where each result requires $\ell - 1$ multiplications (61 K multiplications when $N = 10000$). In contrast, without probability rounding, existing works require 9,026,264 homomorphic operations per word filtering, which is almost 12x more than SNBF. The majority of the multiplications (8,956,264 out of all 9 millions) comes from embedding the fixed-point probability weight into the matched result, where each probability is encoded using a $(52+\delta)$-bit integer (we found δ to be 11 in SNBF). More importantly, the probability rounding results in small bit-width of the comparison results ($\lg(61 \cdot 16)$ <10 bits compared to $(63 + \lg(61))$ bits in [5]). Thus, even if existing works employ our batching technique, we can still batch 6.6x more words per ciphertext (100 compared to 15) for concurrent processing. In combined, our SNBF implementation consumes 1,200x less homomorphic operations than existing works, which directly translates to speed improvement by the same amount even on a CPU implementation.

Overall, the results indicate that the hardware-version of SNBF can filter a word in 0.15 second, 33x faster than the CPU, 398,400x faster than the existing PHE solution [5], and much faster than FHE-based techniques. Furthermore, for the proposed RKM accelerator, we observe nearly 2,000 times power reduction from the software. The time and power reductions combines to a 10^5x reduction in energy when the software and hardware implementations of SNBF are compared, and significant energy reduction compared to existing approaches. Since the average email word count is around 287 in the Enron6 dataset, our SNBF can classify an average-length email within 0.5 s, well in the practical domain.

6 CONCLUSION

In this work, a secure email filter based on the naïve Bayesian filter is proposed. The proposed SNBF relies solely on PHE, and by employing a novel weight-embedding technique, our SNBF requires much less homomorphic computations compared to existing works. In addition to the algorithm, we explore the design space of hardware architecture that truly meets the demand of HE. Through the experiment, our software construction is already faster than existing FHE solutions by three orders of magnitude. The hardware-based SNBF is able to process 100 words in 0.15 s, which we consider to be a practical solution to secure email filtering.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI Grant No. 17H01713, 17J06952, and Grant-in-aid for JSPS Fellow (DC1). The authors also acknowledge support from VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. We thank Xinwen Yao for her support.

REFERENCES

[1] Manuel Barbosa and Pooya Farshim. 2012. Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation.. In *CT-RSA*, Vol. 12. 296–312.
[2] Song Bian, Masayuki Hiromoto, and Takashi Sato. 2017. SCAM: Secured content addressable memory based on homomorphic encryption. In *Proc. DATE*. 984–989.
[3] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. 2004. Public key encryption with keyword search. In *Eurocrypt*, Vol. 3027. 506–522.
[4] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J Wu. 2013. Private database queries using somewhat homomorphic encryption. In *Intl. Conf. Applied Cryptography and Network Security*. 102–118.
[5] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data.. In *NDSS*.
[6] Gary CT Chow, Ken Eguro, Wayne Luk, and Philip Leong. 2010. A Karatsuba-based Montgomery multiplier. In *Intl. Conf. Field Programmable Logic and Applications*. 434–437.
[7] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. 2009. Controlling data in the cloud: outsourcing computation without outsourcing control. In *ACM Cloud Computing Security Workshop*. 85–90.
[8] Yarkin Doröz and Berk Sunar. 2016. Flattening NTRU for Evaluation Key Free Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2016 (2016), 315.
[9] Chelsea P Ferrette. 2007. To Outsource or Not to Outsource: IAs and Emails. *Compliance Reporter* (2007).
[10] Fujitsu Research Institute. 2010. Personal data in the cloud: A global survey of consumer attitudes. http://www.fujitsu.com/downloads/SOL/fai/reports/fujitsu_personal-data-in-the-cloud.pdf. (2010).
[11] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *Symp. Theory of Computing*. ACM, 197–206.
[12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation.. In *NDSS*, Vol. 20. 12.
[13] A. Khedr, G. Gulak, and V. Vaikuntanathan. 2016. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Trans. Comput.* 65, 9 (2016), 2848–2858. DOI: http://dx.doi.org/10.1109/TC.2015.2500576
[14] Ciaran McIvor, Maire McLoone, and John V McCanny. 2004. Modified Montgomery modular multiplication and RSA exponentiation techniques. *IEE Proceedings—Computers and Digital Techniques* 151, 6 (2004), 402–408.
[15] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. 2006. Spam filtering with naive Bayes–which naive Bayes?. In *CEAS*, Vol. 17. 28–69.
[16] Pascal Paillier. 1999. Public-Key Cryptosystems based on Composite Degree Residuosity Classes. In *Intl. Conf. Theory and Applications of Cryptographic Techniques*. 223–238.
[17] Synopsys, Inc. 2013. *Design Compiler I-2013.06*. Synopsys, Inc.