SCAM: Secured Content Addressable Memory Based on Homomorphic Encryption

Song Bian, Masayuki Hiromoto, and Takashi Sato
Deptartment of Communications and Computer Engineering, School of Informatics, Kyoto University
Yoshida-hon-machi, Sakyo, Kyoto 606-8501, Japan
E-mail: paper@easter.kuee.kyoto-u.ac.jp

***Abstract*—We propose an implementation of a secured content addressable memory (SCAM) based on homomorphic encryption (HE), where HE is used to compute the word matching function without the processor knowing what is being searched and the result of matching. By exploiting the shallow logic structure (XNOR followed by AND) of content addressable memory (CAM), we show that SCAM can be implemented with only additive homomorphism, greatly improving the efficiency of the HE algorithm. In the proposed method, the logic of homomorphic XNOR-AND is replaced with homomorphic XOR-OR, requiring only simple additions to be performed on the ciphertext. We also show that our scheme can be implemented by highly parallelizable and simple hardware architecture. Through experiment, we demonstrate that our software implementation is already 403x faster than the fastest known algorithm. With the help of hardware, we can achieve an energy reduction per word match by a factor of 477 million times, making our SCAM scheme much more practical.**

I. Introduction

Private information retrieval (PIR) has been an important area of research over the past few years, as major companies outsource their data to cloud services [1]. The basic setting of PIR is that we have a user who wants to retrieve data from a server, but does not want the server to know what he/she has retrieved. In implementing PIR schemes, homomorphic encryption (HE) is a good theoretical choice, due to HE's ability to directly perform computation on encrypted data [2]. Numerous PIR schemes utilizing HE have been proposed [3]–[5] with optimizations on the underlying HE scheme to minimize the communication overhead between the user and the server. However, PIR query can be complex, and these queries are generally performed on a software database that can be arbitrary in size. This results in complex functions being (homomorphically) evaluated on a large number of entries, and the parameters of the HE scheme needs to be large enough to ensure security. Hence, secured PIR schemes are, in general, impractical. For example, a recently proposed HE-based secure database system requires seven days to retrieve a single row from a 10-record database [5], where the majority of the computational time is spent on the expensive bootstrapping operation.

To accelerate PIR query, we consider a special type of high-performance memory, the content addressable memory (CAM). CAM is a type of memory structure that allows a full parallel search on its entire contents to be performed in a single clock cycle [6]. CAM finds application in networking [7], [8] and database [9] to accelerate the search operations. A regular CAM cell can be as small as nine transistors for binary CAM (BCAM) [10]. This efficiency, however, relies heavily on the assumption of performing the searching function on plaintext without any privacy concerns.

It is a natural question to ask then, that if we can design a secured CAM (SCAM), where a user wants to retrieve the addresses of data that match his/her query from a server, without the server knowing the content of the query. It is clear that an SCAM scheme is a PIR scheme with a slightly different protocol: instead of requesting data that matches a complex condition, SCAM simply queries the addresses of certain data. Concretely, let $\mathbf{x}$ and $\mathbf{y}$ be vectors where each vector element $x_i, y_i \in \{0, 1\}$ represents the i-th bit in the binary numbers x and y, respectively. The objective of a traditional BCAM is to compute

$$f(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{w} \overline{x_i \oplus y_i}, \tag{1}$$

where w is the bit width of x and y. The question here is, if the objective is to implement a BCAM, but *homomorphically*, can we find a scheme that is "efficient," in the sense that retrieving a piece of data from a reasonably large CAM costs less than seven days? Existing research already shows that the answer is yes. Khedr et al. [11] showed a GPU implementation of (1) based on the so called GSW scheme [12] that reports a runtime on the millisecond scale for a simple multi-bit word match. However, since their method is merely a straightforward implementation of the GSW scheme, as later explained in section II-C, inefficient homomorphic multiplication is used to implement the matching function, degrading the performance of the scheme.

In this paper, we propose a novel SCAM scheme and its hardware implementation. Our SCAM targets on realizing a secured binary CAM architecture. We exploit the fact that the circuit depth required for homomorphically matching a word is extremely shallow (essentially two levels of boolean gates), and design a new HE scheme for word matching. It is noted that, although our HE scheme is based on existing research [13], the scheme is designed specifically with hardware in mind. Thus, unlike the mere accelerator implemented in [14], the original scheme with the expensive bootstrapping operation [13], or the aforementioned GSW implementation [11], we avoid double-precision multiplications and FFT operations on the algorithm level to enable an efficient hardware implementation. Thus, although the software implementation of our scheme is already 403x (up to 4,836x with 12 cores) faster than the fastest existing scheme, the hardware can easily obtain the same level of performance as the software with a power reduction of 40,000x, as later shown in section IV. The main contribution of this paper is summarized as follows.

978-3-9815370-8-6/17/$31.00

- **A new application of homomorphic encryption**: We target on a new problem of implementing a secured CAM. Albeit its numerous applications, a secured implementation of CAM has not yet been proposed to the best of our knowledge. In this study, we propose an implementation of SCAM based on HE.
- **A hardware-oriented HE scheme**: We propose an XOR-OR homomorphic gate implementation based on the FHEW scheme to efficiently compute a multi-bit word match. While the original FHEW scheme can only handle a single-bit NAND gate, we make substantial modification to enable a much richer class of gates to be implemented using the scheme.
- **A new hardware design problem**: We explore the hardware architecture for our system. Due to the complexity of the homomorphic evaluation function, the required hardware is larger than the unencrypted implementation by several orders of magnitude (e.g., addition on 16 KB of data to compute a single-bit OR gate). Thus, we face distinct hardware design trade-offs.

The rest of this paper is organized as follows. First, in Section II, we specify notations used throughout this paper, and we give an overview on the preliminary knowledge related to our HE scheme. Second, our proposed SCAM scheme is described in Section III. Third, the efficiency of our implementation based on both numerical estimation and a proof-of-concept ASIC resource utilization is demonstrated in Section IV. Finally, we conclude our paper in Section V.

II. Preliminaries

A. Notations

Throughout the paper, we use $\mathbf{x}$ and $\mathbf{y}$ to represent vectors in $\mathbb{Z}_2^w$, where w is a constant bit width. $\mathbf{x}$ and $\mathbf{y}$ are thus the vector representations of the w-bit number x and y, respectively. We use N to represent the number of lines in the CAM, where each CAM line consists of a w-bit integer in plaintext. In this paper, $\lg x$ is the shorthand for $\log_2 x$.

For the security notations, we will use m for plaintext message, c for ciphertext, s for secret key. Enc and Dec denote the encryption and decryption functions. As explained in later sections, (q, n, χ_σ) is a tuple that represents parameters of the HE scheme with integer ring Z_q, dimension n and a distribution χ with parameter σ.

B. Learning with Errors and Homomorphic Encryption

The integer learning with errors (LWE) problem is first introduced by Regev [15] and later Peikert [16] that respectively provide the quantum and classical reductions for the LWE problem to approximate short vector problems in worst-case lattice, which is known to be hard. Due to the simplicity and flexibility of the problem, numerous cryptosystems base their security on the LWE problem [12], [17], [18].

In this study, we adopt the standard LWE-based HE scheme modified by Ducas and Micciancio [13] named FHEW. The reason why this scheme is adopted will be described in the next section (Section II-C), and we first present the FHEW scheme as a preliminary knowledge. The cryptosystem works as follows. First, we determine a set of parameters (q, n, χ_σ) according to the security level λ we want to achieve, where $q, n \in \mathbb{Z}$, and χ is some distribution with a parameter σ. In a typical LWE setup, χ_σ denotes the discrete Gaussian distribution with a standard deviation σ, and a more detailed definition of this distribution can be found in (6) of [15]. For a integer $t < q$, a message m can be chosen from the message space $\mathbb{Z}_t$. To encrypt the message, we first uniformly draw two vectors $\mathbf{a}, \mathbf{s} \in_R \mathbb{Z}_q^n$, where $\in_R \mathbb{Z}_q^n$ means uniformly randomly sampling n times from the congruence class of integers $\mathbb{Z} \bmod q$. Next, we sample an error $\varepsilon \leftarrow \chi_\sigma$, and ε is a "small" integer (in the sense that $\varepsilon \ll q$). The encryption and decryption functions Enc, Dec then respectively work as follows:

$$b = Enc(m) = \langle \mathbf{a}, \mathbf{s} \rangle + \frac{mq}{t} + \varepsilon \bmod q \in \mathbb{Z}_q, \tag{2}$$

$$m = Dec(b) = \left\lfloor \frac{t}{q}((b - \langle \mathbf{a}, \mathbf{s} \rangle) \bmod q) \right\rceil \bmod t \in \mathbb{Z}_t. \tag{3}$$

Note that q, n, and $\mathbf{a}$ will be published to allow homomorphic evaluation of any functions on the ciphertext, and the ciphertext per se is thus $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$. The above scheme works as long as $|\varepsilon| < q/2t$, since the rounding function $\lfloor \cdot \rceil$ will round off any error less than 1/2 from the original message m. Additive homomorphism is achieved by setting the encryption error $|\varepsilon| < q/4t$. In this way, we can perform an addition of two ciphertexts and still get a ciphertext with error less than $1/2$ during decryption. Multiplicative homomorphism on such classic LWE-based HE scheme is known to be troublesome, where only one multiplication can be performed using the tensor product [17]. Brakerski and Vaikuntanathan extended the multiplication capability of the scheme by adopting the relinearization and modulus switching technique [18], [19] to allow a polynomial number of multiplications to be performed on the ciphertext, albeit the impractical space overheads.

To implement a NAND without multiplicative homomorphism, Ducas and Micciancio observed that, for m_0 and m_1 encrypted as $c_1 = (\mathbf{a}_0, b_0)$ and $c_2 = (\mathbf{a}_1, b_1)$ using the plaintext space $\mathbb{Z}_4$ (when $t = 4$) with error $|\varepsilon| < q/4t = q/16$, $(-\mathbf{a}_0 - \mathbf{a}_1, \frac{5q}{8} - b_0 - b_1)$ is a correct encryption of $\mathrm{NAND}(m_0, m_1)$ in a different plaintext space $\mathbb{Z}_2$ ($t = 2$) with error $|\varepsilon| < q/2t = q/4$. Due to space limitation, we only give a short derivation, and the truth table for (4) is shown in Table I which exactly describe a NAND function.

$$\begin{aligned} &Dec((-\mathbf{a}_0 - \mathbf{a}_1, \frac{5q}{8} - b_0 - b_1)) \\ &= \left\lfloor \frac{2}{q} \left(\frac{5q}{8} - (b_0 + b_1) - (\mathbf{a}_0\mathbf{s} + \mathbf{a}_1\mathbf{s}) \right) \right\rceil \bmod 2 \\ &= \left\lfloor \frac{2}{q} \left(\frac{5q}{8} - \frac{m_0 q}{4} - \frac{m_1 q}{4} \pm \varepsilon_0 \pm \varepsilon_1 \right) \right\rceil \bmod 2 \end{aligned}$$

Since all $|\varepsilon_i| < q/16$, we can combine them and get

$$\left\lfloor \frac{5}{4} - \frac{m_0}{2} - \frac{m_1}{2} \pm 4\varepsilon \right\rceil \bmod 2. \tag{4}$$

C. Comparison of LWE-based HE Schemes

Multiplication is an essential part of homomorphic encryption. Although expensive, multiplication is generally required for fully homomorphism for the design of a universal gate. For example, in the GSW scheme [12], Gentry, Sahai, and Waters show an implementation of homomorphic NAND gate by computing $I_N - C_1 \cdot C_2$, where I and C_i are matrices.

TABLE I
TRUTH TABLE FOR THE NAND GATE

m_0	m_1	$\frac{5}{4} - \frac{m_0}{2} - \frac{m_1}{2} \pm 4\varepsilon$
0	0	$\lfloor \frac{5}{4} \pm 4\varepsilon \rceil = 1$
0	1	$\lfloor \frac{3}{4} \pm 4\varepsilon \rceil = 1$
1	0	$\lfloor \frac{3}{4} \pm 4\varepsilon \rceil = 1$
1	1	$\lfloor \frac{1}{4} \pm 4\varepsilon \rceil = 0$

Khedr et al. take the very same idea and implemented the word matching function $f = \prod_{i=1}^{w} \overline{x_i \oplus y_i}$ [11] as mentioned in Section I. When encrypted under the GSM scheme, this operation can be implemented by $I_n - (C_i - C_j)^2$ for XNOR, and several levels of simple multiplications for AND. Obviously, matrix multiplication is much more expensive than addition, with a best known algorithm runs on a time complexity of $O(n^{2.3727})$ [20]. Khedr et al. reported a speed difference between multiplication and addition of around 17x even on a modern high-performance GPU [11]. Thus, in their scheme, the runtime of the word matching function is dominated by matrix multiplications.

If we can implement a universal gate without the need of multiplication, both the time complexity and the computational resources required can be dramatically reduced. This is the idea behind the FHEW scheme, where Ducas and Micciancio showed an implementation of NAND gate with only additive homomorphism [13]. Unfortunately, a single-bit NAND gate is all the FHEW scheme can do without bootstrapping. The idea of the FHEW scheme is to bootstrap the ciphertext (i.e., refresh the ciphertext to reduce the error level) after each and every homomorphic gate operation. Consequently, a single 2-bit homomorphic NAND gate requires around 0.5 second to evaluate on a modern CPU, where the runtime is dominated by the bootstrapping algorithm. In terms of practical performance, FHEW is much worse than the GSW scheme by orders of magnitude.

A natural question to ask at this point is, if all we want to do is matching words, can we design a better scheme that neither requires multiplication nor bootstrapping. In the next section, we will show how to implement a two-stage complex boolean gate with only additive homomorphism without bootstrapping.

III. PROPOSED SCAM SCHEME

Here, we first present the overall SCAM structure including the communication model in section III-A. Then, the underlying HE scheme is explained in III-B and hardware architecture to implement such SCAM is discussed in section III-C.

A. Two-Round SCAM Protocol

The content addressable memory, as its name suggests, performs a slightly different task compared to PIR. In a typical homomorphic PIR scheme, encrypted addresses (or data) are sent to the server to retrieve the corresponding data [21]. Contrarily, a CAM receives a piece of data, and outputs the address of the CAM line where a word match has occurred. In addition, in a *secured* CAM scheme where the server holds the SCAM and its content, we cannot disclose this address to the server. Thus, we adopt a two-round communication scheme that is known to be optimal [22] as shown in Fig. 1. In the protocol, a user sends the encrypted query c_{request}

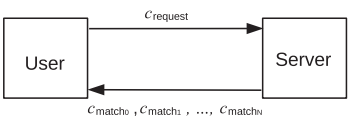

Fig. 1. The two-round communication protocol adopted in this work.

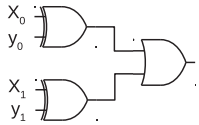

Fig. 2. A gate-level diagram of a 2-bit XOR-OR complex gate.

TABLE II
THE TRUTH TABLE OF A 2-BIT XOR-OR COMPLEX GATE

x_0	y_0	x_1	y_1	XOR-OR	$\widehat{\text{HomXOR-OR}}$
0	0	0	0	0	0
0	0	0	1	1	-1
0	0	1	0	1	1
0	0	1	1	0	0
0	1	0	0	1	-2
0	1	0	1	1	-3
0	1	1	0	1	-1
0	1	1	1	1	-2
1	0	0	0	1	2
1	0	0	1	1	1
1	0	1	0	1	3
1	0	1	1	1	2
1	1	0	0	0	0
1	1	0	1	1	-1
1	1	1	0	1	1
1	1	1	1	0	0

to the server, and the server replies the encrypted matching results $c_{\text{match}_1}, c_{\text{match}_2}, \cdots, c_{\text{match}_N}$, where N is the size of the database, i.e., the number of CAM lines.

B. Homomorphic Encryption Scheme

To implement the function shown in (1), the first problem we have is, while both XNOR and AND are easy to implement individually using the FHEW scheme [13], it is hard to combine them to form an XNOR-AND complex gate. As a workaround, we consider the implementation of an XOR-OR complex gate, and an XNOR-AND gate can be obtained by simply negating the result. Fig. 2 shows a simple 2-bit XOR-OR gate that has four inputs (we name it 2-bit since it compares 2 bits of two target values).

For the design of a homomorphic XOR-OR gate, with only additive homomorphism, we have severe canceling problems. The implementation of a real homomorphic XOR function without reduction modulo q requires multiplication, as we can compute $\text{HomXOR}(\mathbf{c}_{x_i}, \mathbf{c}_{y_i}) = (\mathbf{c}_{x_i} - \mathbf{c}_{y_i})^2$, where $\mathbf{c}_{x_i}$ and $\mathbf{c}_{y_i}$ indicate the ciphertext encrypting x_i and $y_i \in \mathbb{Z}_2$, respectively. Without multiplication, however, we can still "simulate" the real XOR using only the function $\widehat{\text{HomXOR-OR}}$ as follows:

$$\widehat{\text{HomXOR-OR}}(\mathbf{x}, \mathbf{y}) = \sum_{i=0}^{w} (\mathbf{c}_{x_i} - \mathbf{c}_{y_i}). \tag{5}$$

Equation (5), however, introduces an asymmetry in the result of $x_i = 1$, $y_i = 0$ where $\widehat{\text{HomXOR}}$ evaluates to positive, and $x_i = 0$, $y_i = 1$ negative. If we sum the result of two $\widehat{\text{HomXOR}}$ s' (this is precisely the function implementing Fig. 2), canceling happens when a positive number is added to a negative number. For example, consider the input $(x_1, x_0) = (1, 0)$ and $(y_1, y_0) = (0, 1)$, the result should be 1 indicating a mismatch, but $Dec((Enc(1) - Enc(0)) + (Enc(0) - Enc(1))) = 0$ due to canceling.

To counter the canceling problem, we introduce an *encryption constant*, denoted as k_i, to the i-th bit of $\mathbf{x}$ and $\mathbf{y}$. k_i here

is a simple constant holding the value $k_i = 2^i$. We extend the plaintext space of the FHEW scheme from $\mathbb{Z}_4$ to $\mathbb{Z}_t$ where $t = k_{w+1} = 2^{w+1}$, and encrypt each bit of $\mathbf{x}$ and $\mathbf{y}$ using the corresponding encryption constant. The modified version of the encryption and decryption function is then devised as follows.

For each x_i in $\mathbf{x} = [x_w, x_{w-1}, \cdots, x_0]^{\mathrm{T}}$ where $x_i \in \{0, 1\}$,

$$c_{x_i} = Enc(x_i) = (\mathbf{a}, b),$$
$$\text{where} \quad b = \langle \mathbf{a}, \mathbf{s} \rangle + \frac{k_i x_i q}{t} + \varepsilon \bmod q, \quad (6)$$

$$k_i x_i = Dec(c_{x_i}) = \left\lfloor \frac{t}{q}((b - \langle \mathbf{a}, \mathbf{s} \rangle) \bmod q) \right\rceil \bmod t. \quad (7)$$

It is noticed that the decryption function does not recover x_i. Instead, it gives $k_i x_i$. This is acceptable since all we care is if the result is zero (indicating a match) or not (a mismatch). In what follows, we give short proofs for the correctness and security of our scheme.

1) Correctness:

Lemma 1. *Using the encryption and decryption schemes in (6) and (7), the equality relation of* $\mathbf{x}$ *and* $\mathbf{y}$ *is homomorphically evaluated by (5).*

Proof. We prove Lemma 1 by induction.

Base case: For $w = 2$, we have two two-bit numbers $\mathbf{x} = (x_1, x_0)$ and $\mathbf{y} = (y_1, y_0)$. If we decompose (5), we have

$$\widehat{\text{HomXOR-OR}}(\mathbf{x}, \mathbf{y}) = c_{x_1} - c_{y_1} + c_{x_0} - c_{y_0}.$$

Since the encryption function is a simple linear sum, we can isolate the message term and get

$$\frac{q}{t}(2x_1 - 2y_1 + x_0 - y_0) + others. \quad (8)$$

It is noted that the *others* (the $\langle \mathbf{a}_i, \mathbf{s}_i \rangle$ and ε_i term) and the constant $\frac{q}{t}$ in (8) will vanish during the decryption step. Thus, the correctness of the function is illustrated by sketching the truth table of (8), and the result is appended to Table II. A zero-equality test on the decrypted result will show us if $\mathbf{x} = \mathbf{y}$, where 0 indicates a match.

Inductive Step: Suppose $\widehat{\text{HomXOR-OR}}(\mathbf{x}, \mathbf{y})$ correctly evaluates the equality relation between $\mathbf{x}$ and $\mathbf{y}$ where $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^{w-1}$. Adding the w-th bit to $\mathbf{x}$ and $\mathbf{y}$ results in the following equation.

$$\widehat{\text{HomXOR-OR}}((x_w, \mathbf{x}), (y_w, \mathbf{y})) = \mathbf{c}_{x_w} - \mathbf{c}_{y_w} + \sum_{i=0}^{w-1} (\mathbf{c}_{x_i} - \mathbf{c}_{y_i}),$$

where $\sum_{i=0}^{w-1}(\mathbf{c}_{x_i} - \mathbf{c}_{y_i})$ decrypts to 0 only when $\mathbf{x}$ and $\mathbf{y}$ matches. Using the same way to isolate the plaintext as in the base case, we get

$$\frac{q}{t}\left(2^w(x_w - y_w) + \sum_{i=0}^{w-1} 2^i(x_i - y_i)\right) + others. \quad (9)$$

We have two cases here: i) when $\mathbf{x}$ and $\mathbf{y}$ match, and ii) when $\mathbf{x}$ and $\mathbf{y}$ mismatch. In i), the evaluation result depends on if x_w and y_w match; if they do, (9) evaluates to 0, and if they do not, it is either 2^w or -2^w. Thus, (9) is correct for i). For ii), it is observed that the range of the sum in (9) is $S \in [-(2^w - 1), -1] \cup [1, (2^w - 1)]$ where the interval is over integers (notice $\sum_{i=0}^{w-1} 2^i = 2^w - 1$). The evaluation result will always be a mismatch: if x_w and y_w match, the result will be non-zero due to the lower-bit sum. If x_w and y_w does not match, (9) evaluates to $\pm 2^w + S$, and is always non-zero. Hence, (9) is also correct for ii). □

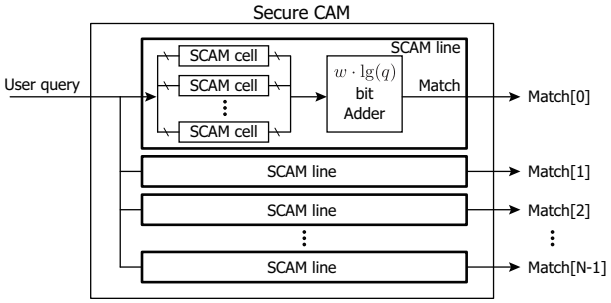

Fig. 3. The general architecture of the proposed SCAM, where a user query is sent to each SCAM line for a match, and the matching result will be sent back to the user. Each SCAM line consists of w SCAM cells that perform bit-wise comparison of the query and the stored content.

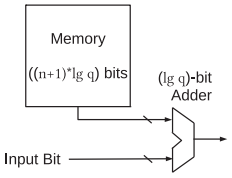

Fig. 4. The structure of an SCAM cell consists of a $n \cdot \lg q$-bit memory cell, a $\lg q$-bit register, and a $\lg q$-bit adder.

2) Security: It is noticed that we did not fundamentally change the encryption and decryption scheme. It is basically the same scheme as FHEW (as the standard LWE-based HE scheme), only with a much larger plaintext space (from $t = 4$ to $t = 2^{w+1}$). The security follows proofs in [13] and [23], where the runtime of the best-known attack against LWE-based cryptosystem almost entirely depends on the root-Hermit factor δ [23]. δ can be obtained from the following formula:

$$\delta = 2^{(\lg^2 \beta)/(4n \lg q)}, \text{ where } \beta = (q/\sigma) \cdot \sqrt{\ln(1/\varepsilon)/\pi}, \quad (10)$$

where $\ln x$ is the natural log. Due to the large size of the plaintext, we need a modulus $q > t$ for correctness, where $t = 2^{w+1}$, and a reasonable choice on q suffice. However, as [23] suggests, larger q induces greater δ that yields a successful attack on the scheme, albeit extremely slowly ($q^{\Theta(1/n)}$ where $n \geq 500$ in the FHEW scheme). Thus, q and n should be adjusted accordingly to ensure the security of the standard LWE-based homomorphic cryptosystem. We give a concrete instantiation of the parameters in section IV.

C. Hardware Architecture

In terms of fully exploiting the amount of parallelism presents in the proposed HE scheme, a GPU-like structure should be adopted. However, for our specific algorithmic design, an ASIC implementation is more efficient for the two facts: 1) adder is the only computational resource required,

and 2) a large quantity of on-chip memory (instead of caches) is required, since an SCAM is still a memory unit.

Here we describe a plausible high-level architecture to implement the proposed SCAM scheme, and leave a more concrete parameter instantiation to Section IV. Fig. 3 demonstrates the overall structure of the proposed SCAM. A *user* inputs w $n \cdot \lg q$-bit ciphertexts that represents w bits of plaintext to the *server*, where homomorphic matching is performed by the server. Each SCAM cell performs homomorphic bit-wise comparison, and the results are homomorphically ORed together using the $w \cdot \lg q$-bit adder tree to produce the match result for an SCAM line. This result, as described in Fig. 1, will be sent back to the user.

Fig. 4 depicts the structure of a single SCAM cell, where a per-bit ciphertext comparison ($\widehat{\text{HomXOR-OR}}$) is performed. Each integer in the ciphertext is in $\mathbb{Z}_q$, so the bit width of the memory as well as the adder needs to be $\lg q$. By setting q to be a power of 2, the modulo q operation becomes free by using an adder with exactly $\lg q$-bit bus width. The memory is responsible for holding the ciphertext of a one-bit plaintext, and the size of the ciphertext is $(n+1) \cdot \lg q$ bits. Note here that by encrypting one of the words to be matched as a negative number in the first place, we do not need to implement a ciphertext subtracter, and a trivial adder should suffice.

Since each SCAM cell produces a vector of $n+1$ $\lg q$-bit integers, a $\lg q$-bit adder tree with $w \cdot (n+1)$ fan-in is required to OR the results of each SCAM cell. To avoid this huge adder tree, we can serialize this part of the computation by summing w $\lg q$-bit integers at a time for $n+1$ cycles. In addition, all $n+1$ additions can be performed in parallel, so this is essentially a design choice to make, where more adder hardware delivers less latency in terms of clock cycles. Additionally, simple pipelining can parallelize the serial addition required in the SCAM line. Thus, we can produce one lg q-bit integer at a time for the comparison result, and the whole comparison process only takes $n+1$ clock cycles.

IV. Experiment

In this section, we present a concrete instantiation of our SCAM scheme, and discuss the hardware performance based on numerical experiment. We will mainly compare our result to the SHIELD scheme proposed in [11], since it is the best known scheme that can be used to implement a SCAM.

A. LWE Parameters and Storage Improvement

Table III summarizes two possible instantiation of our scheme with the corresponding security level (refer to (5.2) in [23] for the equation to calculate the security level). Here, our setup is to permit a 32-bit integer to be encrypted using (6), where a plaintext space of $t = 2^{33}$ is required. Using the proposed parameter, for a single-bit plaintext, we have 5.523 KB ciphertext for medium (80-bit) security, and 6.919 KB ciphertext for high (128-bit) security. Compared to the GSW scheme adopted in [11], for the same 80-bit security level, we reduce the storage for each ciphertext bit by 83 times (from 487 KB to 5.523 KB per bit). Note that our ciphertext size depends on the width of w of the SCAM line, and the ciphertext can be reduced to 440 bytes per bit for high security if a 2-bit comparator is enough for the purpose.

TABLE III
Parameters for the proposed scheme

lg t	lg q	n	σ	Adv. lg ε	δ	Security (λ)
33	42	1052	8	64	1.00658	80 (Medium)
33	42	1318	8	64	1.00525	128 (High)

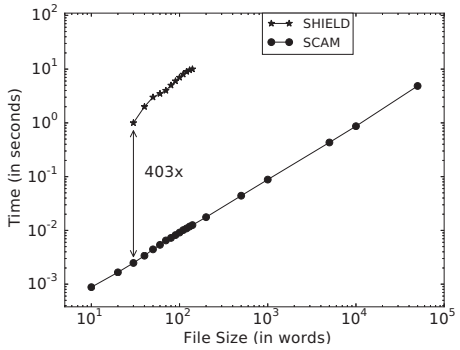

Fig. 5. Runtime comparison between the proposed SCAM and SHIELD [11]. The SCAM is on 128-bit security and SHIELD is of 80-bit security.

B. CPU Implementation

Here we discuss the performance gain of the proposed SCAM scheme implemented on a conventional CPU. We base our implementation on the FHEW scheme that has its source code publicly available [13]. Without the need of expensive bootstrapping operation, our SCAM scheme can be implemented by several hundred lines of code. Our scheme is implemented in the C++ language, and we ran the experiment on an Intel Xeon E5-2630 2.3 GHz processor using a single core with 32 GB of memory (unless otherwise stated).

A CPU implementation of the $\widehat{\text{HomXOR-OR}}$ gate requires $n+1$ integer additions and bit-wise integer AND operation (for modulo q) for a single ciphertext addition. Since there are w bits that need to be XORed and ORed together to produce a comparison result, we need to perform $w+(w-1)$ times ciphertext addition to produce the comparison result. Assuming integer addition and integer AND can be performed in a single CPU clock cycle, it requires $(2 \cdot w-1) \cdot 2(n+1)$ clock cycles to produce a single-line match result. In our setup, this means $(2 \cdot 32-1) \cdot 2 \cdot (1052+1) = 132,678$ (166, 194 for high security) cycles for the CPU to compare one SCAM line. Therefore, on the Xeon processor, in theory, we can perform a one-line comparison in $57.7\,\mu$s (resp. $72.3\,\mu$s), equivalent to a throughput of 17,331 (resp. 13,831) lines per second per core for medium and high security.

In practice, the runtime of the $\widehat{\text{HomXOR-OR}}$ gate with respect to the number of SCAM lines is summarized in Fig. 5. Due to the unoptimized code, the average recorded comparison speed ($91\,\mu$s and $111\,\mu$s per SCAM line for medium and high security) is slightly slower than the theoretical speed. Compared to SHIELD [11], we improved the time complexity from $O(n^{2.3727})$ (multiplication) to $O(n)$ (addition) and space complexity from $O(n^2)$ to $O(n)$. This translates to a speedup of 403x for searching through a file with 30 words assuming that each word can be represented by a 32-bit integer, and up to 4,836x when utilizing all the 12 cores on the CPU. Moreover, this is a comparison between a conventional CPU and a high-performance GPU as in [11]. Asymptotically, our scheme will

TABLE IV
SUMMARY OF THE PERFORMANCE OF A SINGLE SCAM LINE

Power	Area (Gate Count)	Delay
1.205 mW	52,198	9 ns

TABLE V
SUMMARY OF THE PERFORMANCE COMPARISON BETWEEN SHIELD AND CPU SCAM AS WELL AS ASIC SCAM

	SHIELD [11]	SCAM (CPU)	SCAM (ASIC)
Ciphertext Size	487.5 KB	6.8 KB	6.8 KB
Speed	0.033 s	7.58 μs	9.47 μs
Power	165 W	95 W	1.205 mW
Energy	5.445 J	0.72 mJ	11.41 nJ

outperform all the existing methods by orders of magnitude.

C. Hardware Implementation

In this section, we compare the CPU and ASIC implementations of our SCAM scheme. As the homomorphic computation becomes quite trivial in our scheme, an ASIC implementation improves the power performance significantly. We implemented the SCAM line shown in Figs. 3 and 4 using the Verilog language and synthesized the design using a logic-synthesis tool [24] on a commercial 65 nm technology node. Table IV summarizes the power, area and delay of a single SCAM line without the memory cell, and area here is measured by the count of equivalent number of NAND2 gates. With pipelining, we can perform the addition for $\widetilde{\text{HomXOR}}$ and $\widetilde{\text{HomOR}}$ in parallel. Since we compare one pair of $\lg q$-bit integers at a time, we need $n+1$ cycles for a single line-comparison to complete. Thus, ideally, each SCAM line takes 9.47 μs to process a word match for medium security (11.87 μs for high). The speed performance of the ASIC implementation is only 10x faster than the CPU implementation since the CPU runs at a much higher clock frequency. Even with the clock frequency benefit, we only need two ASIC SCAM lines to outperform the CPU, assuming that the CPU can fully utilize its 12 cores. According to Table IV, two lines only cost us 2.4 mW of power. The Xeon CPU requires an average power consumption of 95 W, translating to roughly 40,000x power reduction.

Table V summarize the performance comparison of the SHIELD method in [11], our CPU implementation running full parallel on 12 cores, and our ASIC implementation with a single SCAM line. The speed is measured in per word latency, i.e., the time it takes to evaluate a single word match. Compared to SHIELD, our ASIC implementation reduces the overall energy consumption by approximately 477 million times, essentially making the difference between a "theoretical" and a "practical" implementation.

V. CONCLUSION

We presented a SCAM scheme using a novel homomorphic encryption scheme, and thoroughly evaluated the performance of the scheme on both software and hardware. By squashing out multiplication and bootstrapping from the homomorphic encryption system, we proposed an algorithm that has efficient implementation on hardware. In the experiment, we demonstrate that the software implementation of our scheme is already up to 4,836x faster than the fastest method known, and a hardware implementation can be as fast as the software implementation with a power reduction of 40,000 times.

ACKNOWLEDGMENT

The authors thank Shunmiao Xu for the fruitful discussions. This work was partially supported by JSPS KAKENHI Grant No. 26280014.

REFERENCES

[1] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *Proc. ACM Workshop on Cloud Computing Security*, 2009, pp. 85–90.

[2] V. Vaikuntanathan, "Computing blindfolded: New developments in fully homomorphic encryption," in *Proc. IEEE 52nd Annu. Symp. Foundations of Computer Science*, 2011, pp. 5–16.

[3] X. Yi, M. G. Kaosar, R. Paulet, and E. Bertino, "Single-database private information retrieval from fully homomorphic encryption," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 5, pp. 1125–1134, 2013.

[4] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Proc. Intl. Conf. Applied Cryptography and Network Security*, 2013, pp. 102–118.

[5] Y. Gahi, M. Guennoun, and K. El-Khatib, "A secure database system using homomorphic encryption schemes," *arXiv preprint arXiv:1512.03498*, 2015.

[6] I. Arsovski, T. Chandler, and A. Sheikholeslami, "A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme," *IEEE J. Solid-State Circuits*, vol. 38, no. 1, pp. 155–158, 2003.

[7] F. Yu and R. H. Katz, "Efficient multi-match packet classification with TCAM," in *Proc. 12th Annu. IEEE Symp. High Performance Interconnects*, 2004, pp. 28–34.

[8] A. X. Liu, C. R. Meiners, and E. Torng, "Packet classification using binary content addressable memory," in *Proc. IEEE Conf. Computer Communications*, 2014, pp. 628–636.

[9] B. A. Spinney, "Address lookup in packet data communications link, using hashing and content-addressable memory," 1995, US Patent 5,414,704.

[10] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[11] A. Khedr, G. Gulak, and V. Vaikuntanathan, "SHIELD: Scalable homomorphic implementation of encrypted data-classifiers," *IEEE Trans. Comput.*, vol. 65, no. 9, pp. 2848–2858, 2016.

[12] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Advances in Cryptology*, 2013, pp. 75–92.

[13] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Proc. Annu. Intl. Conf. Theory and Applications of Cryptographic Techniques*, 2015, pp. 617–640.

[14] Y. Doröz, E. Öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1509–1521, 2015.

[15] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, p. 34, 2009.

[16] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem," in *Proc. Annu. ACM Symp. Theory of Computing*, 2009, pp. 333–342.

[17] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Proc. Annu. ACM Symp. Theory of Computing*, 2008, pp. 197–206.

[18] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM J. Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[19] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. Innovations in Theoretical Computer Science Conf.*, 2012, pp. 309–325.

[20] V. V. Williams, "Multiplying matrices faster than Coppersmith-Winograd," in *Proc. Annu. ACM Symp. Theory of Computing*, 2012, pp. 887–898.

[21] C. Gentry, K. A. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs, "Optimizing ORAM and using it efficiently for secure computation," in *Proc. Intl. Symp. Privacy Enhancing Technologies*, 2013, pp. 1–18.

[22] P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key FHE," in *Proc. Annu. Intl. Conf. Theory and Applications of Cryptographic Techniques*, 2016, pp. 735–763.

[23] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Proc. Cryptographers Track at the RSA Conf.*, 2011, pp. 319–339.

[24] *Design Compiler I-2013.06*, Synopsys, Inc.