# Improved Blind Seer System With Constant Communication Rounds

Zhangxiang Hu[*]

June 13, 2017

**Abstract**

Private query brings new challenges to the design of Database Management System (DBMS). A recent work of Blind Seer system introduces a new setup which has an efficient sublinear search for arbitrary boolean query. It splits an index server from server such that client can communicate with index server to retrieve encrypted records. During the query, the server learns nothing about the query.

The original system requires the server to stay online to wait for the client to request for the decryption key. In this work, we design a new protocol for client only communicate with the index server. In addition, the communication rounds between client and index server in the original protocol are linear to the depth of search tree. In our protocol, we show how to reduce it to constant rounds.

## 1 Introduction

In multi-party secure computation, private database querying is an important real-world application. This problem is a generalization of symmetric private information retrieval where clients can make queries consisting of boolean expressions on keyword, and retrieve records that satisfy those keywords in a secure manner. For security, query privacy is one of the most important properties. Roughly speaking, in a database management systems (DBMS), private queries enable a client to retrieve results of its queries without learning anything else about the database and the server learns nothing about the client's queries. Thus, privacy addresses both data and queries. For example, a federal agent may query a hospital for all patients who have

[*]University of Oregon; `huz@cs.uoregon.edu`

had a fever in the last three month. During the query, the federal agent does not want the hospital to know the query he made to avoid panic, and the hospital should not leak any information beyond the records which satisfy the query.

An intuitive way to protect confidentiality is by encrypting the sensitive data. However, performing queries not only becomes more challenging but some important query information, such as access patterns, are leaked to the server. Fortunately, there are some general solutions for the private database query problem in the semi-honest model: the oblivious RAM program [Gol87], Yao's garbled circuit [Yao86], private information retrieval (PIR) [CKGS98] *et al.* . We can also achieve malicious security with extra cost: for example, the *cut-and-choose* technique for garbled circuits [Lin13] [HKE13] and the ORAM method. The problem with these solutions is that they either run in polynomial time or have very expensive basic steps.

## 1.1 Related work

Significant research effort has been spent on query privacy and many new techniques have been presented. [BKL+13] proposes a new privacy definition for data analysis using secure multi-party computation and designs a privacy preserving data sharing solution in federated database environments based on secure MPC. [GHJR14] addresses private database query problem based on the secure-computation using an ORAM architecture, implements a system for symmetric private queries in the semi-honest adversary model, supporting private database access by either index or keyword using modified Path-ORAM protocol and homomorphic encryption scheme. [BGH+13] implements private database queries using somewhat-homomorphic encryption. The idea is to encode the database and the query as polynomials, with the roots of such polynomials indicating the index of result records.

**Blind Seer system.** In recent work, a new type of system [PKV+14] was introduced called the *Blind Seer* (BLoom filter INDex SEarch of Encrypted Results) system. This system can implement efficient sublinear search for arbitrary Boolean queries. Unlike the traditional client-server setting, Blind Seer model has four parties: server $\mathsf{S}$, client $\mathsf{C}$, index server $\mathsf{IS}$, and query check $\mathsf{QC}$. Here party $\mathsf{QC}$ is the external policy enforcement on queries to ensure that queries are valid and legal to meet the specific requirement. Notice that in the real world the server or index server usually play the role of $\mathsf{QC}$. Also, in their paper $\mathsf{IS}$ is split from the server so the system can achieve better privacy-performance trade-offs. However, this setup is based on one crucial assumption: the server $\mathsf{S}$ and index server $\mathsf{IS}$ do not collude with each other. Otherwise, privacy no longer exists. In fact, we assume that in Blind seer system, no two parties are colluding with each other. An overview of the Blind seer system setting is in figure 1.

We no give a high level description of the Blind seer system. The idea behind the system is by traversing search tree. Each leaf in the tree is associated with each database record and each tree node represents a bloom filter. The bloom filter of node $v$ contains all keywords of its descendants and itself. To decide whether the data is in such bloom filter, $\mathsf{C}$ and $\mathsf{IS}$ run secure function evaluation using Yao's garbled circuit [Yao86]. The full system is as follows, we skip all technical details here and just present the high level idea. The technical details will be discussed in the later section.

**1.** Assume there are $n$ records in the database $(r_1, \ldots, r_n)$. The server $\mathsf{S}$ randomly shuffles the records and receives records $(R_1, \ldots, R_n)$. Then $\mathsf{S}$ uses public key encryption system to encrypt the shuffled records.

**2.** $\mathsf{S}$ constructs standard *bloom filter search tree* $T$ for $(R_1, \ldots, R_n)$ and encrypts all bloom filter. $\mathsf{S}$ sends encrypted records and the encrypted bloom filter search tree to the index server $\mathsf{IS}$.

**3.** To make a search query to the database, the client $\mathsf{C}$ and index server $\mathsf{IS}$ run secure function evaluation from the root of the bloom filter search tree by using Yao's garbled circuit. If the circuit outputs TRUE, then do the computation for all its children until we reach the leaf node.

**4.** Index server $\mathsf{IS}$ sends all satisfied leaf nodes to the client $\mathsf{C}$. Notice that these leaf nodes are the associated encrypted records.

**5.** The server sends decryption key to the client. The client decrypts the records and obtains the real output.

We notice that [PKV+14] does not guarantee perfect privacy as we described in section 1. Indeed, in the real world, it would be too expensive and cost to achieve full privacy. We have to make some tradeoff between privacy and efficiency. Thus, to be practical it usually allows some trivial or "unimportant" information to be leaked to achieve better efficiency. But the data and query are still protected. We describe more detail later in section 3.1.

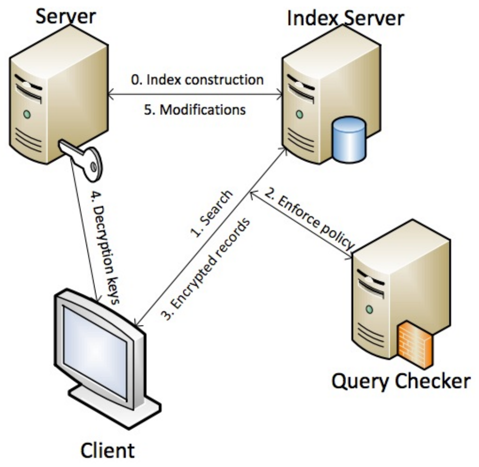

Figure 1: An overview of Blind seer system setting [PKV+14]

## 1.2 Our contribution

We propose a new protocol for the Blind Seer system in which the server is offline and the client only communicate with the index server. More specifically, in the original protocol, the server needs to shuffle and encrypt the records, constructs *bloom filter search tree*, and send them to index server. After that, the server must stay online and wait for the client to request the decryption key. In our setting, the server can be offline after sending the records and search tree to the index server.

Our construction also has a constant number of rounds of communication when client communicating with index server. Unlike the original protocol in which the client and index server need to invoke secure function evaluation and the client needs to send a garbled circuit for each satisfied node in the search tree, our protocol only needs three rounds (assuming client only has temporary storage).

We also give an explicit proof of security that against semi-honest adversaries using a hybrid construction of simulators.

## 2 Preliminaries

Throughout the paper, for an integer $n$, we let $[n] = \{1, 2, \ldots, n\}$. We say a function $\epsilon : \mathbb{N} \to [0, 1]$ is negligible if for any polynomial $p$, there exists a large enough $x'$ such that for all $x > x'$, $\epsilon(x) < 1/p(x)$.

**Parameters:** $l$ is the length of messages and $n$ is the number of messages.
**Private inputs:** the sender has messages $\mathcal{M} = \{m_1, \cdots, m_n\}$ where $m_i \leftarrow \{0,1\}^l$; the receiver has a binary string $b = b_1 \cdots b_n$.

- On input $\mathcal{M}$ from the sender and $b$ from the receiver, give output $\{m_i\}_{b_i=1}$ to the reciver.

Figure 2: Ideal functionality for $k$-out-of-$n$ oblivious transfer.

## 2.1 Secret Sharing

Secret sharing is an elementary cryptographic primitive. Generally speaking, a $t$-out-of-$n$ secret sharing scheme allows a party to split a secret $x$ into $n$ shares. To recover the secret, a collection of $t$ or more shares must be presented. Any collection of less than $t$ shares should leak no information about the secret. One implementation of $t$-out-of-$n$ secret sharing scheme is Shamir's secret sharing scheme using *polynomial interpolation*.

For the special case of $n$-out-of-$n$ secret sharing, there is an efficient scheme using the XOR operation [Sch95]. For example, to split $x$ into $n$ shares such that $x = s_1 \oplus \cdots \oplus s_n$, we can randomly pick $s_2, \cdots, s_n$ and set $s_1 = x \oplus s_2 \oplus \cdots \oplus s_n$.

## 2.2 Oblivious Transfer

Oblivious transfer (OT) is also a very important primitive in cryptography. It involves a sender and a receiver where the sender holds some messages and the client wants to learn a subset of them. An ideal functionality of $k$-out-of-$n$ OT is described in figure 2.

Some possible implementations of $k$-out-of-$n$ OT are [JH09], [Cho12], [WZJ+16].

## 2.3 Garbled Circuit

Yao's garbled circuit was first introduced in [Yao86] as a general solution for two party secure computation. It first expresses a function $f$ as a boolean circuit $C$. For each gate with input wire $w_1, w_2$ and output wire $w_3$, pick two random wire labels $k_0, k_1$ for each wire and associate them with the real input 0 or 1. Then a garbled gate can be constructed according to the truth table. To evaluate the circuit, we recursively evaluate garbled gates and obtain the final output wire labels which are associated with the real output. We refer [LP09] for more details.

Yao's garbled circuit can be efficiently constructed and evaluated in the real world with state-of-art garbling schemes [KS08], [KMR14]. For example, [HEKM11] shows that using garbled circuit, the secure evaluation of AES with 33880 gates can be completed in 0.2 seconds. Even in the malicious model, a recent result [WMK] demonstrates that with security parameter $\kappa = 128$, it takes only 65 ms to evaluate AES and 438 ms for SHA256.

More generally, Yao's garbled circuit can be abstracted and captured by garbling schemes [BHR12] introduced by Bellare, Hoang and Rogaway. They describe a garbing scheme as a five-tuple algorithm $\mathcal{G} = (\mathbf{Gb}, \mathbf{En}, \mathbf{De}, \mathbf{Ev}, \mathbf{ev})$. Let $f$ be the function we want to evaluate,let $x$ be the original input, and $y = f(x)$ the final output. Then $\mathbf{Gb}$ is a randomized algorithm that transforms $f$ into a new triple of functions $(F, e, d)$. $\mathbf{En}$ is an algorithm that takes $(e, x)$ as the input and outputs garbled input $X = \mathbf{En}(e, x)$. $\mathbf{Ev}$ takes input $(F, X)$ and outputs the garbled output $Y = \mathbf{Ev}(F, X)$. $\mathbf{De}$ is an algorithm that transforms garbled output $Y$ to the final output $y = \mathbf{De}(d, Y)$ and we have $\mathbf{De}(d, Y) = \mathbf{ev}(f, x)$. That is, the correctness of the garbling scheme is hold under the condition that $\mathbf{De}(d, \mathbf{Ev}(F, \mathbf{En}(e, x))) = \mathbf{ev}(f, x)$. Specifically, for a garbled circuit, we say $\mathcal{G}$ is a circuit garbling scheme if $\mathbf{ev}$ interprets $f$ as a circuit.

A secure garbling scheme should satisfy some secure properties such as *correctness, privacy, obliviousness and authenticity.* The formal definitions of secure properties can be found in [BHR12].

### 2.4 Bloom filter & Garbled Bloom Filter

A standard bloom filter (BF) [Blo70] is a probabilistic data structure for membership testing. It is a $m$-bit array $\mathcal{B}$ which comes with $k$ independent hash functions $\mathcal{H} = \{h_1, \cdots, h_k\}_{h_i:\{0,1\}^* \to [m]}$.

Let $S = \{x_1, \cdots, x_n\}$ be a set of $n$ elements. To construct a bloom filter $\mathcal{B}_s$, all bits in $\mathcal{B}_s$ are initialized to 0. To add an element $x \in S$ into $\mathcal{B}_s$, we sets $\mathcal{B}_s[h_i(x)] = 1$ for all $1 \leq i \leq k$. To check if an item $x$ is present in $\mathcal{B}_s$, we check whether $\mathcal{B}_s[h_i(x)] = 1$ for all $1 \leq i \leq k$. If any of these bits are 0, then $x$ is not in $S$. Otherwise, we say $x$ is in $S$ with some probability.

It is easy to see that if $x$ is in $S$, then all associated bits must be 1 which means we will never have *false negative*. However, we may have *false positive* such that, for an item $x$ that is not in $S$, all associated bits are set to 1. The probability of false positive is bounded by:

$$\left(1 - (1 - \frac{1}{m})^{kn}\right)^k \approx (1 - e^{-kn/m})^k$$

which is negligible in $k$.

**Garbled Bloom Filter.** Garbled Bloom Filter (GBF) [DCW13] is a variant of the standard BF. GBF supports the same functionality as BF. The only difference is, instead of using a bit array, GBF uses an array of $\lambda$-bit strings. To add an element $x$ to a GBF, we first split $x$ into $k$ shares and store one share in each corresponding location $h_i(x)$. To test whether $x$ is in the bloom filter, we just find its $k$ shares at $h_i(x)$ in the GBF, recover $x'$ by doing a XOR operation of all shares and checking whether $x = x'$.

One issue with GBF here is, for two different elements $x_1$, $x_2$, they can be hashed to the same location. In this case, GBF will reuse the shares in such location. For example, let $s_1 = s_1^1 \oplus s_1^2 \oplus s_1^3$ and share $s_1^1$ is stored in location $j$. When adding element $x_2$ to GBF and $x_2$ also hashes to location $j$, then reuse the share $s_1^1$ and let $x_2 = s_1^1 \oplus s_2^2 \oplus s_2^3$. After adding all elements, generate random strings for locations that have no share and store them in those locations. Reusing shares will not cause security problems, [DCW13] shows that when adding an element to the GBF, the probability that all locations the element hashes to have been occupied by previously added elements is negligible. In other words, the probability that successfully adding $n$ elements to the GBF is

$$1 - p^k \times (1 + O(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{k}}))$$

where $p = 1 - (1 - 1/m)^{k(n-1)}$.

## 3 Technical Overview

We start by describing the detailed protocol of the Blind Seer system. Then we will present a brief discussion of our contribution of how to achieve a server offline model and reduce the communication rounds to a constant number.

### 3.1 The Blind Seer System Paradigm

Now we give a detailed description of the Blind Seer system.

**Bloom Filter Search Tree.** To understand a Bloom Filter Search Tree (BFST), consider figure 3, for a $b$-ary balance tree $T$ with height $\log_b n$, each

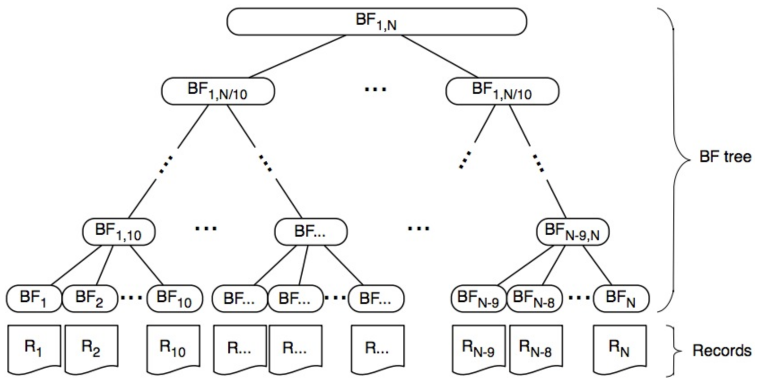

Figure 3: An overview of Blind seer system setting [PKV+14]

leaf node is associated with a database record and each internal node $v$ is associated with a bloom filter $\mathcal{B}_v$. The filter $\mathcal{B}_v$ contains all the keywords from the records that the node $v$ has. For example, if a node contains a record that has Jeff in the fname field, then we set a keyword $e =$ 'fname=Jeff' and insert it to the bloom filter $\mathcal{B}_v$. Notice that a bloom filter $BF_{a,b}$ contains all keywords of records from $R_a$ to $R_b$ and all bloom filters at the same level have the same length depending on the upper bound of number of keywords at that level.

Therefore, for an incoming query with keyword $(e_1, \cdots, e_t)$ wanting to search $(e_1 \wedge, \cdots, \wedge e_t)$, we start from the root of the bloom filter search tree and check if it contains all keywords. If so, we continue checking all of its children. We do this recursively until we reach the leaf nodes and the records are the expected results with all the keywords.

**Detailed Protocol** $\Pi$**.** The functionality consist two phases, initialization and query. Protocol $\Pi$ [PKV+14]:

- Init: The server encrypts the database, shuffles all records and construct the encrypted bloom filter. Then S sends everything to the index server IS. Also, IS will hide the decryption key and shuffle them. When the client ask the server for the decryption keys, S learns nothing about which keys are sent to the client. Let $n$ be the number of records in the database.

  1. The server generates a key pair $(pk, sk)$ for a public-key semi-homomorphic encryption scheme (Gen, Enc, Dec). Then the

server $\mathsf{S}$ randomly shuffles the database and obtains the shuffled database $R = (R_1, \ldots, R_n)$ where $R_i$ is a record. To encrypt the database, for each record $R_i$, $\mathsf{S}$ picks a random string $s_i$, computes $\tilde{s_i} \leftarrow \mathsf{Enc}_{pk}(s_i)$ and set encrypted record $\tilde{R_i} = G(s_i) \oplus R_i$ where $G$ is a pseudorandom generator (PRG).

2. The server constructs a bloom filter as we described above for database $R$. $\mathsf{S}$ picks a random key $k$ for pseudorandom function (PRF) $F$. For each bloom filter $\mathcal{B}_v$, $\mathsf{S}$ computes $\tilde{\mathcal{B}_v} = \mathcal{B}_v \oplus F_k(v)$.
3. The server sends $(pk, (\tilde{s_i}, \tilde{R_i}))$, $i \in [n]$ and the encrypted bloom filter search tree $\mathcal{B}_v$, $v \in T$ to the index server. Also, $\mathsf{S}$ sends $k$ and the bloom filter hash functions $\mathcal{H}$ to the client.
4. The index server picks a random permutation: $\pi : [n] \rightarrow [n]$. For each $i \in [n]$, it picks a random string $r_i$ and computes $\tilde{s}'_{\pi(i)} \leftarrow \tilde{s_i} \cdot \mathsf{Enc}_{pk}(r_i)$. $\mathsf{IS}$ sends all $\tilde{s_i}'$, $i \in [n]$ to the server. $\mathsf{S}$ decrypts $\tilde{s_i}'$ and obtains $s'_i$. Notice that $s'_{\pi(i)} = s_i \cdot r_i$.

- Query: In this phase, the client and the index server will run a secure function evaluation protocol to traverse the tree $T$. At the end, the client obtains the expected encrypted records. Then the client gets the corresponding secret key from the server and decrypts the records to obtain the final output.

  1. Starting from the root, the client constructs a query garbled circuit corresponding to the given SQL query. The client and index server run the circuit to check whether it contains the query keywords. If so, both parties run the secure function evaluation for its all children.
  2. If the search reaches the leaf node $i$, the index server returns $(\pi(i), r_i, \tilde{R_i})$ to the client.
  3. The client sends $\pi(i)$ to the server and $\mathsf{S}$ returns $s'_{\pi(i)}$. Since $s'_{\pi(i)} = s_i r_i$ and the client receives $r_i$ from $\mathsf{IS}$, it can compute $s_i$ and decrypt $\tilde{R_i}$ to obtain the plain record.

One thing that needs to be mentioned here is that the server sends the encrypted bloom filter to the index server and associated key $k$ to the client. Thus, when client and index runs the SFE protocol, they need to decrypt the bloom filter first. Since we use state-of-art Yao's garbled circuit to implement the SFE protocol, it can be achieved by hardcoding key $k$ into the circuit and let the index server just use the encrypted bloom filter as input. The circuit will decrypt the bloom filter during the evaluation.

**Leaked information.** The allowed information leakage includes:

1. The size of the database (*i.e.*the number of records) are leaked to client $\mathsf{C}$ and index server $\mathsf{IS}$.
2. The client $\mathsf{C}$ knows the traversal path of the bloom filter search tree.
3. Anything about the garbled circuit can be leaked (*i.e.* topology of the circuit).
4. Index server $\mathsf{IS}$ knows the bloom filter indices.

## 3.2 Sever offline model

Now we give a high level overview of the sever offline model where the client only communicates with the index server to make private queries.

**Construct GBF** In [PKV+14], the server encrypts records and BF search tree and sends them to the index server, also sends the PRF key $k$ and hash functions $\mathcal{H}$ to the client. After preparing the record decryption keys $s_i'$ with index server, the server needs to stay online to wait for the request of $s_i'$ from client and send it back to client. Our observation here is that we can use a secret sharing scheme to recover the original records rather than using a public key encryption system.

Consider how $\Pi$ encrypts record $R_i$ using a pseudo-one time pad, it generates a random string $s$ and computes encrypted $\tilde{R_i}$ using PRG $G$:

$$\tilde{R_i} = G(s_i) \oplus R_i$$

Instead of encrypting the original record, we secret sharing $R_i$ and use pseudo-random function to encrypt each share, then constructs a GBF to store encrypted shares according to its keywords. In our protocol, the construction of the BF search tree is the same as in protocol $\Pi$ except for the leaf nodes. When the server constructs the bloom filter $\mathcal{B}_v$ for each leaf node, we also construct a corresponding $GBF_v$ to recover the original record. In particular, for each keyword $e$ in record $R_i$, secret sharing $R_i$ into $\eta$ shares where $\eta$ is the number of hash function that we use to construct BF.

$$R_i = s_1 \oplus s_2 \oplus \cdots \oplus s_\eta$$

The encryption of each share is also mapped into $\eta$ index numbers and we store shares in each location at $h_i(e)$. Let $\mathsf{PRF}$ be a pseudo-random function and $\mathsf{sd}$ be a seed to a PRF. Suppose a share $s_i$ is mapped to location $j$, the encryption is $\tilde{s_i} = \mathsf{PRF}(\mathsf{sd}, v\|j) \oplus s_i$ and stores $\tilde{s_i}$ at location $j$.

**Recover the original record** The client and index server run the secure function evaluation using garbled circuit for $\mathcal{B}_i$. If $\mathcal{B}_i$ contains keyword $e$, the client calls hash functions $\mathcal{H}$ on input $e$ to compute the corresponding indices of $\mathcal{B}_i$ for record $R_i$ [1]. Then client and sever invoke a $\eta - out - of - m$ oblivious transfer, with input $\eta$ indices from the client and $m$ strings from the index server. At the end, the client receives $\eta$ encrypted shares $\{\tilde{s_i}\}_{i\in[\eta]}$, decrypts each share using $\mathsf{PRF}(\mathsf{sd}, v\|j) \oplus \tilde{s_i}$ and recover the original record.

## 3.3 Constant Communication Rounds

In the naive blind seer system, the client and index server need to run secure function evaluation at each level using a garbled circuit. The Yao's garbled circuit has a privacy guaranteed and efficient implemented. Start from the root of BF search tree, for each internal node $\mathcal{B}_v$, if it contains the query keywords, then the circuit outputs all of its children of $\mathcal{B}_v$. We recursively search those children to check if they contain the query keywords. For the leaf node, then the circuit outputs index $v$.

It is easy to see that the number of communication rounds between client and index server in this phase is sublinear to the depth of the BF search tree since we need to generate garbled circuits for BF at each level. To reduce the communication rounds, our construction is inspired by the idea from the recent TWORAM technique [GMP16]. The general idea is that for the search tree in an ORAM, we can hardcode bucket information, input wire labels for the next circuit and other necessary parameters in the garbled circuits. Thus, the output of the circuit includes the next node to be evaluated along with its garbled wire labels.

In our protocol, if the current node is not a leaf, we make the circuit output all nodes whose parent contains the query keywords. If it is a leaf node, output node ID and output wire labels [2] to index server. Index server sends output wire labels to the client and the client calls the decoding algorithm to obtain the plaintext.

Figure 4 is a formal description of the garbled circuit. The inputs to such circuit include keyword statement $e$, garbled wire labels $\mathsf{Label}_{prev}$ from its parent (from client if it is root), bloom filter $\mathcal{B}_v$ and its ID $v$. The outputs of the circuit are all of its children that will be evaluated next along with garbled input wire labels. Also, we need to hardcode $\mathsf{kb}$ which will be

[1] If we search for more than one keyword, such as $e_1$ and $e_2$, the client only needs one of them to recover the record

[2] The output wire labels of leaf node also contains node ID, which indicates that the BF contains the query keyword. Otherwise, wire labels can just indicate $\perp$.

**Inputs:** $(e, \mathsf{Label}_{prev}, \mathcal{B}_v, v)$
**Outputs:** $\{u\}_{u\in \text{children of } v}$ and $\mathsf{Label}_{next}$ of $u$
**Hardcode:** BF decryption key $\mathsf{kb}$, $\mathsf{Label}$ for all of its children

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

- $v$ is not a leaf node. **Return** $(\{u\}_{u\in \text{children of } v}, \mathsf{Label}_u)$ if $\mathcal{B}_v$ contains $e$. If $\mathcal{B}_v$ does not contain $e$, **Return** $\perp$.
- $v$ is a leaf node. **Return** $\mathsf{outputLabels}$.

Figure 4: Formal description of circuit $\mathcal{C}_v$ for BF $\mathcal{B}_v$

used to decrypt $\mathcal{B}_v$ and all garbled wire labels $\mathsf{Label}$ which will be used as garbled inputs for the circuit at the next level. Notice we use $\mathsf{outputLabels}$ to represent the output wire labels for the circuit of the leaf nodes.

## 4 Main Construction

In this section, we present our main construction, which is a semi-honest secure protocol for the functionality described in [PKV+14].

### 4.1 Protocol notations

We use $e$ to denote the query statement and $N$ be the number of records. Let the length of each record be $\lambda$ and the size of GBF is $\ell$.

### 4.2 Detailed protocol

Now we present the full protocol $\pi$. We refer to the client as $\mathsf{C}$, the index server as $\mathsf{IS}$ and the server as $\mathsf{S}$.

**Input of $\mathsf{S}$:** Database records $D$ of size $N$, hash functions $\mathcal{H} = \{h_i : \{0,1\}^* \to [\ell]\}_{i\in[\eta]}$.

**Server:**

1. On input $D = \{D_1, \cdots D_N\}$ and $\mathcal{H}$, $\mathsf{S}$ shuffles the records. Let $R = \{R_1, \cdots, R_N\}$ be the shuffled records.
2. $\mathsf{S}$ constructs BF search tree $T$ as in protocol $\Pi$ for shuffled records $R$. Let $\mathcal{B}_{leaf} = \{\mathcal{B}_1, \cdots, \mathcal{B}_l\}$ be all leaf nodes in $T$. $\mathsf{S}$ randomly chooses a key $kp$ for a pseudo-random function $F$. For each bloom filter $\mathcal{B}_v$ in each node $v$ of $T$, encrypt $\mathcal{B}_v$ as $\tilde{B_v} = \mathcal{B}_v \oplus F(kp, v)$.

3. $\mathsf{S}$ randomly chooses a seed $\mathsf{sd}$ for $\mathsf{PRF}$. For each leaf node $\mathcal{B}_t \in \mathcal{B}_{leaf}$, construct a corresponding garbled bloom filter $\mathcal{GB}_t$ as in section 3.2:
   - For each keyword $e$ in record $R_i$, use XOR secret sharing scheme: $R_i = s_1 \oplus \cdots \oplus s_\eta$. Each share corresponds to a location in $\mathcal{GB}_t$, i.e., $s_i$ corresponds to location $j = h_i(e)$.
   - $\mathsf{S}$ encrypts each share as $\tilde{s_i} \leftarrow \mathsf{PRF}(\mathsf{sd}, t\|j) \oplus s_i$ and stores $\tilde{s_i}$ at location $j$ in $\mathcal{GB}_t$, $\mathcal{GB}_t[j] = \tilde{s_i}$.
   - After $\mathsf{S}$ inserts all shares, if there exists some location $j$ in $\mathcal{GB}_i$ is empty, fill it with a random string: $\mathcal{GB}_t[j] \leftarrow_R \{0,1\}^\lambda$
4. Finally, $\mathsf{S}$ sends $(\{\mathcal{B}_v\}_{v\in T}, \{\mathcal{GB}_t\}_{t\in[l]})$ to the index server and $(kp, \mathsf{sd}, \mathsf{topo}_T)$ to the client.

**Record Retrieval:**

1. On input $\mathsf{topo}_T$ and query statement $e$, the client invokes a garbling scheme as we described in section 2.3 to construct the garbled circuit for each $\mathcal{B}_v$ in $T$. The hardcoded parameters are defined in section 3.3. $\mathsf{C}$ sends all circuits in topological order. Also, $\mathsf{C}$ sends input wire labels of the root circuit to $\mathsf{IS}$.
2. $\mathsf{IS}$ evaluates the root circuit: $(\{u\}_{u\in\text{children of root}}, \mathsf{Label}) \leftarrow \mathbf{Ev}(C_{root}, \mathsf{Label}_{root})$. If it outputs $\perp$, send $\perp$ to $\mathsf{C}$. Otherwise, traverse the BF search tree and recursively evaluate each $u_i \in \{u\}_{u\in\text{children of root}}$.
3. If $\mathsf{IS}$ obtains $\mathsf{outputLabels}$, it sends $\{\mathsf{outputLabels}_v\}_{v\in[l]}$ to $\mathsf{C}$. Otherwise send $\perp$.
4. $\mathsf{C}$ calls $\mathbf{De}(\mathsf{outputLabels})$ and obtains $\{v\}_{v\in[l]}$.
5. $\mathsf{C}$ computes $\mathcal{H}(e) = \{h_i(e)\}_{i\in\eta}$. For each $v$, $\mathsf{C}$ and $\mathsf{IS}$ invokes $\eta$-out-of-$\ell$ OT, with input $\{h_i(e)\}_{i\in\eta}$ from $\mathsf{C}$ and $\mathcal{GB}_v$ from $\mathsf{IS}$. $\mathsf{C}$ receives encrypted shares $\{\tilde{s_i}\}_{i\in[\ell]}$. $\mathsf{C}$ then decrypts each $s_i = \mathsf{PRF}(\mathsf{sd}, v\|j) \oplus \tilde{s_i}$ where $j \in \{h_i(e)\}_{i\in\eta}$ and recovers the output record $R_i = \bigoplus_{i\in\eta} s_i$.

**Other discussion.** Our protocol needs to construct garbled circuits for all BFs in $T$, even if some circuits will not be used during the traversal of the BF search tree. This is because we can not have the pre-knowledge of the size of query statement $e$. Assume for queries that have return records, in the worst case, we construct $N - (1 + b \cdot \lceil \log_b N \rceil)$ extra circuits where $b$ indicates the search tree is a $b$-ary tree with height $\log_b N$.

Consider the situation that a client has a bunch of queries that can be sorted by the size of query statements. For those circuits that are not consumed during the evaluation, we can reuse them for the next query that has the same statement size. However, due to the security nature of garbled circuit, it can be used only once. In other words, once the index server evaluates the circuit, the client needs to refresh that circuit for the next use. To solve this problem, the client needs to download all circuits that are evaluated, decode the hardcoded value, update the garbled circuits and send them back to index server. Notice that the refresh procedure causes extra communication rounds between the client and index server.

Another possible improvement from recent work [HKK+14] addresses the situation that when we want to evaluate the same circuit multiple times, both in parallel and sequentially. It uses LEGO-based cut-and-choose technique to obtain lower amortized overhead in the multiple-execution setting even in the malicious model.

## 4.3 Security Analysis

We now analysis the security of our protocol. But before we prove the security against semi-honest adversary, first we need to specify the leakage information for each party. The leakage profile is described in 5.

**Leakage to S.** S has all records and does not communicate with C or IS after sending all messages to IS. Therefore, nothing is leaked to S.
**Leakage to C.** C needs to refresh all used circuits so the traversal pattern of BF search tree is leaked.
**Leakage to IS.** IS learns some access pattern after multiple queries. Also, IS learns whatever the garbled circuits leak during secure function evaluation.

Figure 5: Leakage profile $\Phi$ of protocol $\pi$

Now we prove protocol $\pi$ is secure against semi-honest adversary under the leakage profile.

**Definition 1.** *Let $f = (f_1, f_2)$ be a deterministic function. We say that protocol $\pi$ securely computes function $f$ in the presence of static semi-honest adversaries if there exists probabilistic polynomial-time algorithms $S_1$ and $S_2$ such that:*

$$\{S_1(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_1^{\pi}(x, y)\}_{x,y}$$
$$\{S_2(y, f_2(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_2^{\pi}(x, y)\}_{x,y}$$

In other words, whatever can be computed by an adversary in the protocol can be simulated by a probabilistic polynomial time simulator from the party's input and output only. Notice that in our protocol $\mathsf{S}$ is offline after sending all messages to $\mathsf{IS}$. Thus, we can consider out protocol as a two-party secure computation between $\mathsf{C}$ and $\mathsf{IS}$.

**Theorem 2.** *The protocol $\pi$ implements the Blind Seer DBMS in the presence of static semi-honest adversaries under the leakage profile $\Phi$.*

*Proof.* We describe two simulators $S_1$, $S_2$, w.r.t $\mathsf{C}$, $\mathsf{IS}$, to simulate the view of each party.

**Client is corrupted.** Client's view consists of PRF key $(kp, \mathsf{sd})$, GBF indices, output wire labels, the view in the oblivious transfer protocols, and the corresponding encrypted shares.

We start by describing simulator $S_1$: On input $(e, R_e)$ where $R_e$ denotes the desired records corresponding to the query statement $e$, $S_1$ uniformly chooses random $(kp', \mathsf{sd}')$, secret sharing $R_e = s'_1 \oplus, \cdots, \oplus s'_\eta$ [3]. Then, $S_1$ randomly chooses $t'$, $(j'_1, \cdots, j'_\eta)$ and encrypt each share $\tilde{s_i}' \leftarrow \mathsf{PRF}(\mathsf{sd}', t' \| j') \oplus s'_i$. Also, $S_1$ generates all garbled circuits and let $S_1^{OT}$ be the simulator to obtains $\mathsf{C}$'s view in oblivious transfer. We guarantee that $S_1^{OT}$ exists because the security of the oblivious transfer protocol. In other words, $S_1$ can simulate client's view in the oblivious transfer without knowing $\mathsf{IS}$'s input. By the correctness of garbled circuit, $\mathsf{C}$ obtains the final output $v$, except only with negligible probability. Notice that $\mathsf{C}$ receives $\mathsf{outputLabels}$ rather than the real output $v$, but the decoding of these distributions of $\mathsf{outputLabels}$ is equal to $v$. So we can generate $\mathsf{outputLabels}'$ from the garbled circuits that are generated from $S_1$. Also, $S_1$ call hash function $\mathcal{H}(e)$ which concludes our description of $S_1$:

$$(e, R_e, kp', \mathsf{sd}', \{\tilde{s_i}'\}_{i \in [\eta]}, S_1^{OT}(e, R_e), \mathcal{H}(e), \mathsf{outputLabels}'\})$$

We now prove that

$$\{S_1(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_1^\pi(x, y)\}_{x,y}$$

in a hybrid model [4].

[3]For simplicity, we assume the final result contains only one record.

[4]In the proof, we omit the random tape $r_C$ that is used for $S_1$ to generate the garbled circuits

- $\mathfrak{H}_0$ : Simulator plays exactly like an honest client and all ideal functionalities except that $S_1$ generates its own randomness of $kp'$ and $\mathsf{sd}'$. It is easy to see that this hybrid is identical to the real interaction with $\pi$.

- $\mathfrak{H}_1$ : Same as $\mathfrak{H}_1$ except that call its own secret sharing scheme and encrypt each share to obtain $\{\tilde{s_i}'\}_{i\in[\eta]}$. We have $\mathfrak{H}_1 \approx \mathfrak{H}_0$ by the security of the secret sharing scheme and PRF, the adversary can distinguish the distributions only with negligible probability.

- $\mathfrak{H}_2$ : $S_1$ generates all garbled circuits and obtains $\mathsf{outputLabels}$' from the decoding of $v$. By the correctness and authenticity of garbled circuits, we have $\mathfrak{H}_2 \approx \mathfrak{H}_1$

- $\mathfrak{H}_3$ : Since $S_1$ has all garbled circuits, $S_1$ replace the real view in oblivious transfer in protocol $\pi$ by the simulated view of $S_1^{OT}(e, R_e)$. And we claim that $\mathfrak{H}_3 \approx \mathfrak{H}_2$ because the security of oblivious transfer.

Hence, the simulator in $\mathfrak{H}_3$ is our final simulator $S_1$ and the proof of this case is concluded.

**Index server is corrupted.** In this case, we also construct a simulator $S_2$ that generates the view of $\mathsf{IS}$ in protocol $\pi$.

The view of $\mathsf{IS}$ consists messages from $\mathsf{S}$ and messages from $\mathsf{C}$. It receives $(\{\mathcal{B}_v\}_{v\in T}, \{\mathcal{GB}_t\}_{t\in[l]})$. However, these messages does not leak any information, thanks to the security of PRF and secret sharing scheme. In other words, we can consider the messages between $\mathsf{S}$ and $\mathsf{IS}$ as initialization phase. In the following, we only construct $S_2$ that simulates the messages that $\mathsf{IS}$ receives from $\mathsf{C}$.

First observe the view of $\mathsf{IS}$. It consists garbled circuits, garbled wire labels for all inputs, and the view in the oblivious transfer protocol.

Now we describing the simulator $S_2$: On input $(\{\mathcal{B}_v\}_{v\in T}, \{\mathcal{GB}_t\}_{t\in[l]})$, $S_2$ needs to simulates all garbled circuits. However, $S_2$ can not construct garbled circuits honestly since it does not know the real input from $\mathsf{C}$. Therefore, $S_2$ construct "fake" garbled circuits that always evaluate to the correct outputs. This can be achieved by letting the garbled tables in which all ciphertexts are the encryption of the same key. During the evaluation, no matter what keys $\mathsf{IS}$ receives, it always evaluate to the same output wire. As this is done in most prior work, we skip the details here. The construction details of "fake" garbled circuit and the simulation of garbled input wire labels refer to [LP04]. For the the view of $\mathsf{IS}$ in the oblivious transfer protocol, as in the case that $\mathsf{C}$ is corrupted, the security properties of oblivious transfer protocol guarantee that there exists a simulator $S_2^{OT}$ that can simulate such view. Therefore, the formal description of $S_2$ is as follows:

$$(GC', \mathsf{Label}', S_2^{OT}(\{\mathcal{B}_v\}_{v\in T}, \{\mathcal{GB}_t\}_{t\in[l]}))$$

And we now prove that

$$\{S_2(y, f_2(x,y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_2^{\pi}(x,y)\}_{x,y}$$

- $\mathfrak{H}_0$ : $S_2$ plays the role of an honest $\mathsf{IS}$ as in the real protocol $\pi$. All messages are from $\mathsf{C}$.
- $\mathfrak{H}_1$ : $\mathfrak{H}_1$ is the same as $\mathfrak{H}_0$ except that, rather than letting $\mathsf{C}$ generates all garbled circuits honestly, $S_2$ generates "fake" garbled circuits $GC'$ as we described above. Also, $S_2$ generates $\mathsf{Label}$' which are from $GC'$. Due to the correctness and obliviousness of garbled circuits, we have $\mathfrak{H}_1 \approx \mathfrak{H}_0$.
- $\mathfrak{H}_2$ : In $\mathfrak{H}_2$, $S_2$ replace the real view in oblivious transfer in protocol $\pi$ by the simulated view of $S_2^{OT}(\{\mathcal{B}_v\}_{v\in T}, \{\mathcal{GB}_t\}_{t\in[l]})$. And we claim that $\mathfrak{H}_2 \approx \mathfrak{H}_1$ because the security of oblivious transfer.

In $\mathfrak{H}_2$, we have our simulator $S_2$ that can construct the view of $\mathsf{IS}$ without knowing $\mathsf{C}$'s real input and the proof of this case is concluded.

□

## References

[BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. Cryptology ePrint Archive, Report 2013/422, 2013. http://eprint.iacr.org/2013/422.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. Cryptology ePrint Archive, Report 2012/265, 2012. http://eprint.iacr.org/2012/265.

[BKL+13] Dan Bogdanov, Liina Kamm, Sven Laur, Pille Pruulmann-Vengerfeldt, Riivo Talviste, and Jan Willemson. Privacy-preserving statistical data analysis on federated databases, 2013. http://research.cyber.ee/~jan/publ/privacypreservingdatabases.pdf.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[Cho12] Jue-Sam Chou. A novel k-out-of-n oblivious transfer protocol from bilinear pairing. *Adv. MultiMedia*, 2012:3:1–3:9, January 2012.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 789–800. ACM Press, November 2013.

[GHJR14] Craig Gentry, Shai Halevi, Charanjit Jutla, and Mariana Raykova. Private database access with HE-over-ORAM architecture. Cryptology ePrint Archive, Report 2014/345, 2014. http://eprint.iacr.org/2014/345.

[GMP16] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. *TWORAM: Efficient Oblivious RAM in Two Rounds with Applications to Searchable Encryption*, pages 563–592. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In Alfred Aho, editor, *19th ACM STOC*, pages 182–194. ACM Press, May 1987.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[HKE13] Yan Huang, Jonathan Katz, and Dave Evans. Efficient secure two-party computation using symmetric cut-and-choose. Cryptology ePrint Archive, Report 2013/081, 2013. http://eprint.iacr.org/2013/081.

[HKK$^+$14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. *Amortizing Garbled Circuits*, pages 458–475. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[JH09] Ashwin Jain and C. Hari. A new efficient k-out-of-n oblivious transfer protocol. *CoRR*, abs/0909.2852, 2009.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, August 2014.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. Cryptology ePrint Archive, Report 2013/079, 2013. http://eprint.iacr.org/2013/079.

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. http://eprint.iacr.org/2004/175.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[PKV+14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 359–374, Washington, DC, USA, 2014. IEEE Computer Society.

[Sch95] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., New York, NY, USA, 1995.

[WMK] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. Faster two-party computation secure against malicious adversaries in the single-execution setting. Technical report, Cryptology ePrint Archive, Report 2016/762, 2016. http://eprint. iacr. org/2016/762.

[WZJ$^+$16] Xiaochao Wei, Chuan Zhao, Han Jiang, Qiuliang Xu, and Hao Wang. *Practical Server-Aided k-out-of-n Oblivious Transfer Protocol*, pages 261–277. Springer International Publishing, Cham, 2016.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.