Towards Secure and Verifiable Database-driven Spectrum Sharing

Zhili Chen[1], Lin Chen[2], Hong Zhong[1]

[1]School of Computer Science and Technology, Anhui University, 230601 Hefei, China
[2]Lab. Recherche Informatique (LRI-CNRS UMR 8623), Univ. Paris-Sud, 91405 Orsay, France
Email: zlchen@ahu.edu.cn, chen@lri.fr, zhongh@mail.ustc.edu.cn

***Abstract*—Database-driven spectrum access is regarded as an effective spectrum redistribution mechanism. However, dialoguing with the spectrum database requires both primary and secondary users to reveal their sensitive data to the spectrum database manager (SDM), leading to serious privacy concerns. In this paper, we show that the SDM can perform database operations (both updates and queries) without knowing any information about the users' sensitive inputs and the database contents, by combining garbled circuits and secret sharing. Our design uses data-oblivious sorting networks to leverage parallelism of query operations, yielding an efficient query algorithm. We further combine secure computations with authentication techniques to get a verification mechanism for correctness checking. As far as we know, our proposal is the first secure and verifiable database-driven spectrum sharing scheme protecting both PUs' and SUs' privacies. Finally, we fully implement our system, and demonstrate that even on commodity PC, our implementation suffers mild performance overhead.**

I. Introduction

The ever increasing spectrum demand for emerging wireless applications and the unbalanced utilization of radio spectrum resource have prompted spectrum redistribution, where unlicensed secondary users (SUs) use or buy idle spectrum from licensed primary users (PUs). For spectrum redistribution, two ways are applied to determine channel availability: spectrum sensing and white space database. In the former, an SU finds an available channel by listening and analyzing the PU's signal in the channel, while in the latter, an SU queries a database to get spectrum availability information (SAI) at its location. It has been shown that database-driven spectrum access usually leads to more efficient spectrum utilization over spectrum sensing, due to the large margins in incumbent detection thresholds in spectrum sensing imposed by regulation authorities [1]. Therefore, FCC recently designated the white space database as a requisite for cognitive radio devices [2].

Despite its effectiveness in spectrum redistribution, database-driven spectrum access also faces new security threats [3], as summarized in the following.

- **PUs' Operational Privacy Threats**. In order to update its channel states, each PU should provide its operational information (e.g. transmitter ID, location, antenna parameter, power, time of operation, etc [3]) to the spectrum database; this may expose the PUs' sensitive information to the spectrum database and may bring security vulnerability if the spectrum database manager is not a trusted party. PUs' operational privacy threats have become even more important and urgent due to the recent calls in the United States by Federal Communications Commission (FCC) for sharing federal government (including military) spectrum in the 3.5 GHz band with non-government systems [4].
- **SUs' Location Privacy Threats**. Before picking a channel to use, each SU should query spectrum availability information from the spectrum database with its geographic location; this may lead to location privacy disclosure of SUs in case of intrusted spectrum database manager. Location information is normally regarded as a user's sensitive information, which can also be used to infer the user's other sensitive information, e.g., health condition, lifestyle and so on. The location privacy threats have been widely acknowledged in the literature, e.g., [5][6][7][8].

Furthermore, if the database manager is malicious, he may not perform database updates correctly, or he may return outdated or false query results. Therefore, it is important to investigate how we can build a database-driven spectrum sharing framework under which both PUs and SUs do not reveal their inputs in the clear and the correctness of database operations is verifiable, which is the focus of this paper.

There exist a number of approaches for protecting users' location privacy in traditional location based services, such as k-anonymity approach, or collaborative location privacy protection [5][6][7]. However, these approaches either requires a trusted server or incur extra overhead in user cooperation, and hence cannot be directly applied in database-driven spectrum sharing. Recently, a scheme called PriSpectrum is proposed in [9] to protect the location privacy of SUs in database-driven spectrum sharing. However, as far as we know, there is no existing work that addresses the problem of the operational privacy of PUs and the correctness verification of database operations in this context.

Motivated by the above arguments, in this paper we devise a secure and verifiable database-driven spectrum sharing scheme, protecting both PUs' operational privacy and SUs' location privacy while ensuring correctness verification of database operations. It is not a priori clear whether spectrum sharing can be performed practically in such a manner that both PUs' and SUs' privacies are protected, and the correctness of database operations is verified. There are three main challenges associated with this task. First, to address the privacy concerns raised above, spectrum sharing should be performed without the spectrum database manager ever learning the PUs' operational information and the SUs' location information, and even the spectrum database. This requirement is key, since any leakage of such information could lead to serious privacy disclosure. Second, such a secure algorithm ought to be efficient, and scale gracefully with the number of operations. Third, an appropriate verification mechanism should be conceived to defend against malicious attacks, while does not damage the running efficiency much. The privacy requirements imply that our spectrum sharing algorithm ought to be data-oblivious: its execution ought to not depend on the user input. Both efficiency and verification requirements mean that the secure algorithm should be elaborately designed.

Our main contributions are articulated as follows.

- *Secure and verifiable protocol:* We design a secure protocol for database-driven spectrum sharing that meets all the above requirements: privacy, efficiency and verifiability. The resulted protocol is hybrid, combing garbled circuits with secret sharing. We then prove that our protocol is cryptographically secure against semi-honest adversaries, and demonstrate that it also defends against several malicious attacks.
- *Data-oblivious algorithm:* We propose and use in our design an efficient data-oblivious database query algorithm yielding a complexity $O(M \log M)$, where M is the number of query operations. This is with $\log M$ factor of query operations in the clear, and achieved by using sorting networks, leveraging parallelism in operations.
- *Correctness verification:* We combine secure computations and MAC techniques to design a correctness verification mechanism for both update and query operations. The proposed verification can defend against various malicious attacks tampering the spectrum database or returning false query results.
- *Experimental evaluation:* We fully implement our secure database-driven spectrum sharing system, and carry out extensive experiments to evaluate its performance. Experimental results demonstrate that our scheme is practical in term of computation and communication performance.

The remainder of this paper is organized as follows. Section II and III provide problem formulation and technical preliminaries. We present in detail our secure and verifiable database-driven spectrum sharing protocol in Section IV, and prove its security in both semi-honest and malicious setting in Section V. In Section VI, we implement our protocol, and evaluate the performance in terms of computation and communication overhead. Section VII briefly reviews related work. Finally, the paper is concluded in Section VIII.

II. System Model and Problem Formulation

A. System Model

We consider a generic database-driven spectrum sharing framework consisting of the following three entities:

- **Primary users (PUs)**: licensed users with exclusive transmission right.
- **Secondary users (SUs)**: unlicensed users seeking spectrum resource unused or leased by PUs.
- **Spectrum database manager (SDM)**: agent managing the spectrum database and making the spectrum allocation decision to SUs based on the spectrum availability information from the database. The SDM can be a spectrum broker or a decision making agent on a base station of a cognitive radio cellular network.

The network terrain is divided into non-overlapping $n \times n$ squares denoted by $\mathcal{S} = \{s_{ij}\}_{n\times n}$, where s_{ij} denotes the square (i, j). The spectrum availability information (SAI) of the network is stored in the spectrum database, which is represented by a $n \times n$ matrix $\mathcal{M} = \{m_{ij}\}_{n\times n}$, where m_{ij} indicates the SAI of the square s_{ij}. More specifically, m_{ij} is a K-element vector $\{\epsilon_k^{ij}\}_K$ with $1 \leq k \leq K$, where ϵ_k^{ij} denotes the SAI of channel k in square s_{ij}. Given square s_{ij} and channel k, ϵ_k^{ij} can be a single-bit value indicating whether channel k is occupied (1) or not (0), or a multi-bit value indicating the signal level of channel k. Without loss of generality, we apply the former for simplicity in this paper.

We denote the number of PUs and SUs by N_p and N_s, and assume that each PU k provides a channel k for simplicity (it is simple to extend to the case with multiple channels), i.e. $N_p = K$. This implies that we regard transmitter ID as public knowledge. The spectrum access and sharing is performed in rounds. At the beginning of each round, the following actions are performed:

- Each PU decides whether or not to use his channel during the coming round, computes its service contour (the geographical range where the channel cannot be reused) using its operational information (e.g. location, antenna parameter, power, etc.), and sends his update request to the SDM. We denote the update request of PU k by $(k, \mathcal{C}_k)$, where $\mathcal{C}_k = \{c_{ij}\}_{n\times n}$ is an $n \times n$ bit matrix with all elements being 0 (null update) or describing the service contour of channel k (update), where $c_{ij} = 1$ means that square s_{ij} overlaps with the service contour and $c_{ij} = 0$ otherwise.
- Any SU wishing to use a particular channel (or several channels) for the coming round sends a query request containing its location (i.e. its located square) to the SDM.
- The SDM updates the spectrum database based on update requests submitted by PUs and queries in the database given the query requests submitted by SUs.

B. Security Goals and Threat Models

We introduce a new entity crypto-service provider (CSP) to cooperate with SDM to carry out two-party secure computations (as illustrated in Fig. 1), and aim at designing a secure and verifiable spectrum sharing scheme as long as SDM and CSP do not collude with each other.

We firstly focus on the *semi-honest* (a.k.a. *honest but curious*) threat model where SDM and CSP follow the protocol we develop but may analyze protocol transcripts to infer additional information. The proposed protocol should not leak any illegitimate information to any entity. By entities we refer to PUs, SUs, SDM and CSP. By illegitimate information we refer to any sensitive information related to other agent. Specifically, the following security requirements should be satisfied:

- No other entity can learn anything about the operational information except the public knowledge transmitter ID,

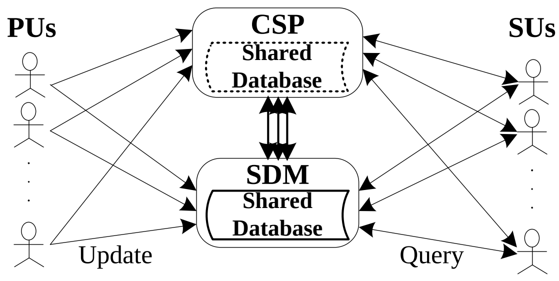


Fig. 1. Secure and verifiable spectrum sharing architecture: inputs, outputs and database are secretly shared between SDM and CSP.

of any PU;
- No other entity can learn anything about the location and query result of any SU;
- No entity can learn anything about the spectrum database except that each PU knows its current channel state;
- Any external entity cannot learn any information about the spectrum sharing.

We then also consider the malicious threat model where an attacker can deviate from the protocol arbitrarily, and require that the correctness of all update and query processing should be well verified in the presence of malicious SDM or CSP.

III. Technical Preliminaries

A. *Garbled Circuits*

Garbled circuits (a.k.a Yao's protocol) [10][11] is a generic method for secure two-party computation. We give a brief overview of garbled circuits via an illustrative example in which two parties securely compute a function $f(x_1, x_2)$ (represented by a boolean circuit) in the presence of semi-honest adversaries. Party i ($i \in \{1, 2\}$) holds an input v_i. Each party privately provides its input, and both parties cooperatively run a garbled circuit evaluating $f(v_1, v_2)$ so that no party learns more than what is revealed from the output. The protocol requires that both parties do not collude with each other.

The core idea of garbled circuits relies in the circuit encoding. One party (as garbler) associates two random cryptographic keys, $K^0_{w_i}$ and $K^1_{w_i}$, to each wire w_i of the boolean circuit computing f, corresponding to the bit-values $b_i = 0$ and $b_i = 1$. For each binary gate g with input wires w_i and w_j, and output wire w_k, the garbler computes the four ciphertexts (e.g., the AND gate as illustrated in Fig. 2)

$$\mathrm{Enc}_{(K^{b_i}_{w_i}, K^{b_j}_{w_j})}(K^{g(b_i,b_j)}_{w_k}) \text{ for } b_i, b_j \in \{0, 1\}.$$

The above four randomly ordered ciphertexts define the garbled gate g. Composed of a series of garbled gates, a garbled circuit can be securely evaluated by the other party (as evaluator).

In our context, all inputs are secretly shared, and finally the garbled output and its decoding information are held by evaluator and garbler, respectively. Continuing with the previous example, let party 1 as *garbler* holding input shares $[v_1]_1$ and $[v_2]_1$ and party 2 as *evaluator* holding $[v_1]_2$ and $[v_2]_2$, our garbled circuit protocol can be outlined as follows.

b_i	b_j	$g(b_i, b_j)$	Garbled value
0	0	0	$\mathrm{Enc}_{(K^0_{w_i}, K^0_{w_j})}(K^0_{w_k})$
0	1	0	$\mathrm{Enc}_{(K^0_{w_i}, K^1_{w_j})}(K^0_{w_k})$
1	0	0	$\mathrm{Enc}_{(K^1_{w_i}, K^0_{w_j})}(K^0_{w_k})$
1	1	1	$\mathrm{Enc}_{(K^1_{w_i}, K^1_{w_j})}(K^1_{w_k})$

Fig. 2. Illustration of a garbled AND gate (i.e., $g(b_i, b_j) = b_i \wedge b_j$).

- **Garbler:** generates a garbled circuit computing f, garbles its input shares $[v_1]_1$ and $[v_2]_1$, and sends the garbled circuit, garbled $[v_1]_1$ and $[v_2]_1$ to the evaluator, while holds the output decoding information itself.
- **Evaluator:** upon receiving the garbled circuit, garbled $[v_1]_1$ and $[v_2]_1$, the evaluator computes the garbled inputs from the received garbled values and its input shares $[v_1]_2$ and $[v_2]_2$, and then feeds the garbled inputs to the garbled circuit to obtain the garbled output.

Note that with inputs secretly shared, the garbled circuit protocol can work without a 1-out-of-2 oblivious transfer protocol [12].

B. *XOR Secret Sharing*

In this paper, we apply the XOR secret sharing [13] to split a bit vector into two shares. Formally, the XOR secret sharing scheme in the case of two sharing parties can be described as Definition 1.

Definition 1 (XOR Secret Sharing). *A bit vector $x \in \{0, 1\}^K$ can be dispersed into 2 shares as follows:*

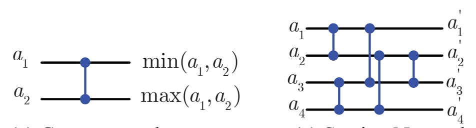
- *The first share $[x]_1$ is randomly chosen in $\{0, 1\}^K$;*
- *The second share is $[x]_2 \in \{0, 1\}^K$ with $[x]_1 \oplus [x]_2 = x$, and thus $[x]_2 = [x]_1 \oplus x$.*

The definition has a very straightforward security property: if an adversary obtains merely a share of x, he gets nothing about x except the bit length of x.

C. *Sorting Networks*

A sorting network is a boolean circuit which sorts an input array $(a_1, a_2, ..., a_n)$ into a monotonically increasing array $(a'_1, a'_2, ..., a'_n)$. The main building block of sorting networks is *compare-and-swap* circuits, which are binary operators taking as input a pair (a_1, a_2), and outputting the sorted pair (a'_1, a'_2) with $a'_1 = \min(a_1, a_2)$ and $a'_2 = \max(a_1, a_2)$.

Sorting networks can be used as data-oblivious sorting algorithms for security purposes. The efficiency of a sorting network can be measured by its complexity, i.e. the total number of compare-and-swap circuits it includes. A theoretically optimal sorting network is the well-known AKS network that achieves a complexity of $O(n \log n)$. However, being an important theoretical discovery, the AKS network has no practical application due to a large constant. Therefore, in this paper, we use the AKS network for theoretical analysis, while use a practically efficient sorting network, the Batcher's odd-even



(a) Compare-and-swap (a) Sorting Network

Fig. 3. Odd-even merge sorting network [15]: $n = 4$

merge network [14][15] in the experiments. Batcher's sorting network can be depicted as shown in Alg. 1. A compare-and-swap circuit and an odd-even merge sorting network for $n = 4$ are illustrated in Fig. 3.

Algorithm 1 Odd-even Merge Sorting Network Algorithms

(a) oeSort(a, inc)

```
Input: $a = \{a_1, a_2, \cdots, a_n\}$, $inc = 0$ or 1
Output: $c$ ($inc = 0$: non-increasing, $inc = 1$: non-decreasing)
1: if $n > 1$ then
2:     $m = \frac{n+1}{2}$;
3:     oeSort($a[1..m], inc$);
4:     oeSort($a[m+1..n], inc$);
5:     $c$ =oeMerge($a[1..m], a[m+1..n], inc$);
6: end if
       return $c$;
```

(b) oeMerge(a, b, inc)

```
Input: $a = \{a_1, a_2, \cdots, a_m\}$, $b = \{b_1, b_2, \cdots, b_n\}$, $inc = 0$
    or 1
Output: $c$
1: if ($m == 0$ || $n == 0$) then
2:     $c = \langle a, b \rangle$;
3: else if ($m == 1$ && $n == 1$) then
4:     swap($a_1, b_1, inc == 1$ ? $[a_1 > b_1]$ : $[a_1 < b_1]$);
5:     $c = \langle a, b \rangle$;
6: else
7:     $c[1, 3, \cdots]$ =oeMerge($a[1, 3, \cdots], b[1, 3, \cdots], inc$);
8:     $c[2, 4, \cdots]$ =oeMerge($a[2, 4, \cdots], b[2, 4, \cdots], inc$);
9:     for ($i = 2$; $i < n$; $i += 2$) do
10:        $bl = (inc == 1)$ ? $[c[i] > c[i+1]]$ : $[c[i] < c[i+1]]$;
11:        swap($c[i], c[i+1], bl$);
12:    end for
13: end if
       return $c$;
```

D. Message Authentication Codes

Message authentication codes (MACs) are used to verify the integrity of a message. MACs ensure that data received are exactly as sent by, i.e., there is no modification, insertion, deletion, or replay. Formally, A MAC is a triple of efficient algorithms (G, S, V) satisfying:

- **Key-generator** G gives the key k on input 1^n, i.e., $k \leftarrow G(1^n)$, where n is the security parameter.
- **Signing algorithm** S outputs a tag t given the key k and the input string x, i.e., $t \leftarrow S(k, x)$.
- **Verifying algorithm** V outputs accepted or rejected on inputs including the key k, the string x and the tag t, i.e., $V(k, x, t) = \text{accepted}|\text{rejected}$.

TABLE I. NOTATIONS FOR OUR PROTOCOL

$\mathcal{M}$	the SAI matrix, $\mathcal{M} = \{m_{ij}\}_{n \times n}$
m_{ij}	the SAI of square s_{ij}, $m_{ij} = \{\epsilon_k^{ij}\}_K$
ϵ_k^{ij}	the SAI of channel k in square s_{ij}
$\mathcal{C}_k$	the service contour of the kth spectrum channel
K	the number of PUs or spectrum channels
N_s	the number of SUs or query requests
l_u	query location of SU u, $l_u = (i_u - 1)n + j_u$ indexing square (i_u, j_u)
d_u	decoding information of SU u
$G(.)$	the garbling function
$[x]_s, [x]_c$	SDM's and CSP's shares of x, where $[x]_s \oplus [x]_c = x$

Algorithms G, S and V must satisfy

$$Pr[V(k, x, S(k, x)) = \text{accepted} | k \leftarrow G(1^n)] = 1$$

The way to develop an efficient MAC is to use cryptographic hash functions which generally execute faster than other cryptographic primitives, and the resulting MAC is called HMAC. An HMAC can be expressed as

$$\text{HMAC}(K, M) = H[(K^+ \oplus \text{opad}) || H[(K^+ \oplus \text{ipad}) || M]]$$

where

- H is the embedded hash function (e.g., MD5, SHA-1).
- M is the message input to HMAC (including the padding specified in the embedded hash function).
- K is the secret key, with a recommended length not less than the security parameter n.
- K^+ is K padded with zeros on the left so that the result is b bits in length.
- ipad = 00110110 (36 in hexadecimal) repeated b/8 times.
- opad = 01011100 (5C in hexadecimal) repeated b/8 times.

In the above, b is the number of bits in a block of the hash function H. We will use HMAC-MD5 to detect the malicious modifications in our experiments.

IV. SECURE AND VERIFIABLE DESIGN

In this section, we first give the overview of our secure database-driven spectrum sharing protocol. Then, the detailed design of the protocol is presented step by step. Finally, we augment the proposed protocol with correctness verification, defending against some common malicious attacks.

A. Protocol Overview

The protocol overview is shown in Fig. 4, where both SDM and CSP perform the secure computations cooperatively. Briefly speaking, all sensitive data are secretly shared, update operations are performed in the secret sharing form, and query operations are performed using garbled circuits. Notations for the protocol can be summarized in Tab. I.

More specifically, in our protocol, we apply a hybrid approach by combining garbled circuits and secret sharing. We let both the SDM and the CSP secretly share the spectrum available information (SAI) matrix $\mathcal{M}$ using the XOR secret sharing. Each PU (resp. SU) splits its update (resp. query) request into two shares in the same way, and sends one share

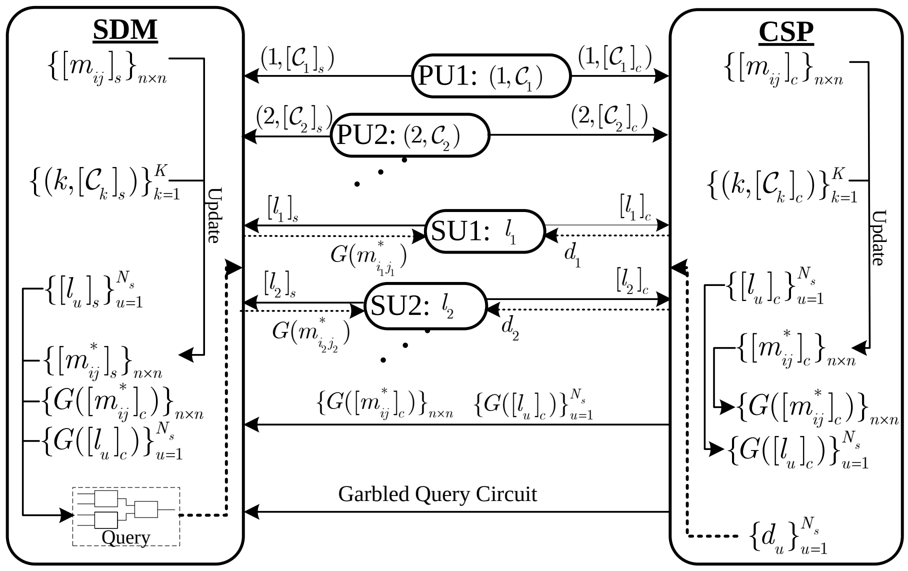

SDM
CSP
PU1: $(1, \mathcal{C}_1)$
PU2: $(2, \mathcal{C}_2)$
SU1: l_1
SU2: l_2
$(1,[\mathcal{C}_1]_s)$
$(1,[\mathcal{C}_1]_c)$
$(2,[\mathcal{C}_2]_s)$
$(2,[\mathcal{C}_2]_c)$
$[l_1]_s$
$[l_1]_c$
$[l_2]_s$
$[l_2]_c$
$G(m^*_{i_1j_1})$
$G(m^*_{i_2j_2})$
d_1
d_2
$\{[m_{ij}]_s\}_{n\times n}$
$\{(k,[\mathcal{C}_k]_s)\}_{k=1}^K$
Update
$\{[l_u]_s\}_{u=1}^{N_s}$
$\{[m^*_{ij}]_s\}_{n\times n}$
$\{G([m^*_{ij}]_c)\}_{n\times n}$
$\{G([l_u]_c)\}_{u=1}^{N_s}$
Query
$\{[m_{ij}]_c\}_{n\times n}$
$\{(k,[\mathcal{C}_k]_c)\}_{k=1}^K$
$\{[l_u]_c\}_{u=1}^{N_s}$
$\{[m^*_{ij}]_c\}_{n\times n}$
$\{G([m^*_{ij}]_c)\}_{n\times n}$
$\{G([l_u]_c)\}_{u=1}^{N_s}$
$\{G([m^*_{ij}]_c)\}_{n\times n}$ $\{G([l_u]_c)\}_{u=1}^{N_s}$
Garbled Query Circuit
$\{d_u\}_{u=1}^{N_s}$

Fig. 4. The overview of our secure database-driven spectrum sharing protocol: SAI matrix $\mathcal{M}$ is initially shared between SDM and CSP. Users submit one share of requests to SDM and the other to CSP. Update operations are performed independently over its respective share by SDM or CSP, while query operations are performed cooperatively by both of them using garbled circuits. Finally, each garbled query result and its decoding are sent to the corresponding SU.

to SDM and the other to CSP. To perform update operations, either SDM or CSP can simply XOR up its shares of $\mathcal{M}$ and update requests; To perform query operations, both SDM and CSP apply the garbled circuit protocol, resulting that SDM holds the garbled query results while CSP holds the decoding information. The appropriate garbled query result and its corresponding decoding information are then sent to each SU, who can thus obtain its plain query result. The protocol for each round of spectrum sharing is illustrated in Protocol 2. Here, two points need to be emphasized as follows.

- Each PU needs to submit its update request at the beginning of each round, even though it has nothing to update (in this case, it submits a null update request). This protects the privacy of operation time for each PU.
- Each SU submits its query request when necessary, since only the privacy of location for each SU is concerned. Thus, number of participant SU in each round is probably different.

In the above, how to perform the update processing and how to construct the query circuit are detailed in Sec. IV-B and Sec. IV-D, respectively. Additionally, we assume that all communication channels are authenticated (which can be implemented using regular cryptographic ways), and mainly focus on the secure computations during the spectrum sharing.

B. Update Processing

In this phase, SDM and CSP update their respective shares of $\mathcal{M}$, independently. More specifically, each of them can simply XOR up its share of SAI matrix $\mathcal{M}$ with its share of all update requests from PUs. To make this workable, each PU has to prepare its update request properly.

Let us illustrate how to prepare an update request by a simple example in Fig. 5. Given a value v shared by SDM (holding share $[v]_s$) and CSP (holding share $[v]_c$), if a PU wants to update v to get v_1, it splits v_1 into $[v_1]_s$ and $[v_1]_c$,

	SDM	**CSP**
Old value:	$[v]_s$	$[v]_c$
Update request:	$[v_1]_s$	$v \oplus [v_1]_c$
Updated value:	$[v]_s \oplus [v_1]_s$	$[v]_c \oplus v \oplus [v_1]_c$

Fig. 5. A simple example for update request.

and sends $[v_1]_s$ to SDM and $v \oplus [v_1]_c$ to CSP. Then, SDM updates its share $[v]_s$ by computing $[v^*]_s = [v]_s \oplus [v_1]_s$, and CSP updates its share $[v]_c$ by computing $[v^*]_c = [v]_c \oplus v \oplus [v_1]_c$. It is obvious that $[v^*]_s \oplus [v^*]_c = v_1$, namely, v is successfully updated to be v_1. This ensures the independent update of v shared between SDM and CSP.

Each PU can prepare its update request in exactly the same way as the above example. Specifically, if PU k wants to update its SAI $\{\epsilon_k^{ij}\}_{n\times n}$ shared between SDM and CSP to get $\{\epsilon_{k*}^{ij}\}_{n\times n}$, it can prepare a share of update request

$$[\mathcal{C}_k]_s = \{[\mathcal{C}_k^{ij}]_s\}_{n\times n} = \{[\epsilon_{k*}^{ij}]_s\}_{n\times n}$$

for SDM and the other share

$$[\mathcal{C}_k]_c = \{[\mathcal{C}_k^{ij}]_c\}_{n\times n} = \{\epsilon_k^{ij} \oplus [\epsilon_{k*}^{ij}]_c\}_{n\times n}$$

for CSP.

Having all update requests prepared as above, the update algorithm is straightforward, as shown in Alg. 3. This algorithm does not apply any cryptographical tools, but updates spectrum database directly in the secret sharing form. It is run by both SDM and CSP independently, without any interaction. Thus, the algorithm incurs very slight performance overhead compared to the underlying algorithm without any security guarantees.

The complexity of the update algorithm is $O(KN)$ where $N = n^2$ is the number of squares, which is the same as that of the underlying unsecure algorithm.

```
Protocol 2 Secure Database-driven Spectrum Sharing Protocol
Input: A boolean circuit $Q$ computing query operations; the secretly shared spectrum database $\mathcal{M} = \{m_{ij}\}_{n\times n}$, with one share $\{[m_{ij}]_s\}_{n\times n}$ held by SDM and the other share $\{[m_{ij}]_c\}_{n\times n}$ held by CSP; update requests from PUs and query requests from SUs.
Output: The updated matrix $\mathcal{M}^*$ secretly shared by SDM and CSP; the query result for each SU.

Phase 1: Secret Sharing and Submission
1: Each PU $k$: splits its update request $(k, \mathcal{C}_k)$ into two shares $(k, [\mathcal{C}_k]_s)$ and $(k, [\mathcal{C}_k]_c)$, and sends the former to SDM and the latter to CSP.
2: Each SU $u$: splits its query request $l_u$ into two shares $[l_u]_s$ and $[l_u]_c$, and sends the former to SDM and the latter to CSP.

Phase 2: Update Processing
3: SDM: takes as input $\{[m_{ij}]_s\}_{n\times n}$ and $\{(k, [\mathcal{C}_k]_s)\}_{k=1}^K$, and gets its updated share of $\mathcal{M}$, $\{[m^*_{ij}]_s\}_{n\times n}$, independently.
4: CSP: takes as input $\{[m_{ij}]_c\}_{n\times n}$ and $\{(k, [\mathcal{C}_k]_c)\}_{k=1}^K$, and gets its updated share of $\mathcal{M}$, $\{[m^*_{ij}]_c\}_{n\times n}$, independently.

Phase 3: Query Processing
5: CSP: takes as input the query circuit, $\{[m^*_{ij}]_c\}_{n\times n}$ and $\{[l_u]_c\}_{u=1}^{N_s}$, generates a garbled query circuit $G(Q)$, garbled values $\{G([m^*_{ij}]_c)\}_{n\times n}$ and $\{G([l_u]_c)\}_{u=1}^{N_s}$, and decoding information $\{d_u\}_{u=1}^{N_s}$, and sends $G(Q)$ and the garbled values to CSP.
6: SDM: upon receiving the garbled circuit and values from CSP, takes as input $\{[m^*_{ij}]_s\}_{n\times n}$, $\{[l_u]_s\}_{u=1}^{N_s}$, $\{G([m^*_{ij}]_c)\}_{n\times n}$ and $\{G([l_u]_c)\}_{u=1}^{N_s}$ to compute the garbled input to query, $\{G(m^*_{ij})\}_{n\times n}$ and $\{G(l_u)\}_{u=1}^{N_s}$. With the garbled input and garbled circuit, CSP computes the garbled query results $\{G(m^*_{i_u j_u})\}_{u=1}^{N_s}$.
7: SDM: sends each SU $u$ the garbled query result $G(m^*_{i_u j_u})$.
8: CSP: sends each SU $u$ the decoding information $d_u$.
9: Each SU $u$: decodes $G(m^*_{i_u j_u})$ with $d_u$ to get $m^*_{i_u j_u}$.
```

```
Algorithm 3 Update Processing
Input: Matrix $\mathcal{M} = \{[m_{ij}]_x\}_{n\times n}$, $\{(k, [\mathcal{C}_k]_x)\}_{k=1}^K$, where "x" can be "s" or "c".
Output: Updated matrix $\mathcal{M}^* = \{[m^*_{ij}]_x\}_{n\times n}$
1: for $i = 1 \dots n$ do
2:     for $j = 1 \dots n$ do
3:         $[t_{ij}]_x = ||_{k=1}^K [\mathcal{C}_k^{ij}]_x$;
4:         $[m^*_{ij}]_x = [m_{ij}]_x \oplus [t_{ij}]_x$;
5:     end for
6: end for
```

C. Query Circuit: A Naive Design

We now design the query circuit. We first present a naive design which, in spite of its naivety, allows us to get more insights on the problem, based on which we present our efficient design.

Consider the task of querying an entry in a matrix with N entries given its location (i.e. indexes). We can do this in time $O(1)$ using random access memory. However, when considering a secure version of this task in which the location of entry should be protected, we need at least $\Omega(N)$ time. The reason is that every entry should be touched, otherwise some information about the location can be inferred by analyzing which entries are accessed.

Along this line of thinking, we can design a naive circuit querying an entry of the matrix as follows. The circuit compares the location of the entry queried with the location of each entry in the matrix, and then the query operation is performed based on the comparison results. For multiple queries, the same circuit can be applied repeatedly using different locations as required. Using this circuit, the secure matrix query can be achieved by running the corresponding garbled circuit protocol between non-colluding SDM and CSP.

However, as the number of queries from SUs increases, this design will become very inefficient. Given the number of SUs' query requests N_s, the time complexity of this naive circuit is $O(N \cdot N_s)$. In practice, it is probable that $N_s \sim \Omega(N)$, and the complexity becomes $\Omega(N^2)$. Even when N is normally large (e.g. $N = 100 \times 100 = 10000$), the complexity $\Omega(N^2)$ is quit prohibitive for secure computations.

The inefficiency of the naive design arises from its inability to batch multiple matrix accesses and to amortize the cost for every single access, which motivates our design of a more efficient query circuit.

D. Query Circuit: Our Efficient Design

Inspired by the applications of sorting networks in [15], [16], we design our efficient batch matrix query using a sorting network. Different from [15], which used sorting networks to leverage sparsity in the data, we make use of sorting networks to take advantage of the parallelism in multiple oblivious random accesses to a database.

Specifically, we write the SAI matrix $\mathcal{M} = \{m_{ij}\}_{n\times n}$ as an array $\mathcal{M}_a = \{a_i\}_{i=1}^N$, where $N = n \times n$ and $a_{(i-1)\times n+j} = m_{ij}$. The query circuit is described in Algorithm 4. Our main idea is to store SUs' queries together with the SAI matrix $\mathcal{M}$ in an array. Through appropriate sorting operations, the query tuples can be placed immediately after the SAI matrix entry with which they share an index; a linear pass through the data allows the computation of query.

```
Algorithm 4 Query Circuit
Input: Tuples $\{(i, a_i)\}_{i=1}^N$, $\{(s_i, j_i, v_i)\}_{i=1}^{N_s}$
1: Initialize tuple array $S$
2: Sort $S$ with respect to rows 2 and 3
3: Query Computing (left-to-right pass):
       $$s_{4,k} = s_{4,k} \cdot (1 - s_{3,k}) \oplus s_{4,k-1} \cdot s_{3,k} \quad (1)$$
   for $k = 2 \dots N + N_s$
4: Sort $S$ with respect to rows 3 and 1
5: Output item $s_{4,k}$, $k = 1 \dots N_s$
```

In the following we first describe the algorithm in detail and then discuss its implementation as a circuit.

Initialization. The algorithm takes as input the array $\mathcal{M}_a$ and SUs' queries $\{(s_i, j_i)\}_{i=1}^{N_s}$. These input data constitute an $(N + N_s)$ array of tuples S. The first N tuples of S store the entries of $\mathcal{M}_a$, while the remaining N_s tuples store SUs' queries. More specifically, for each entry a_i in $\mathcal{M}_a$, the algorithm constructs a matrix tuple $(0, i, 0, a_i)$, where a_i is a K-element vector indicating if each channel is used. For each query request i, the algorithm constructs a query tuple $(s_i, j_i, 1, v_i)$, where s_i is the ID of the SU, j_i is the index of the square the SU wishes to query, and v_i is a K-element vector storing the query result. The resulting initial tuple array S is as follows:

$$\begin{pmatrix} 1: & 0 & 0 & \dots & 0 & s_1 & s_2 & \dots & s_{N_s} \\ 2: & 1 & 2 & \dots & N & j_1 & j_2 & \dots & j_{N_s} \\ 3: & 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ 4: & a_1 & a_2 & \dots & a_N & v_1 & v_2 & \dots & v_{N_s} \end{pmatrix}$$

We denote by $s_{l,k}$ the l-th entry of the k-th tuple. These entries serve the following roles:

$s_{1,k}$:	the IDs of SUs.
$s_{2,k}$:	the indexes of squares concerned.
$s_{3,k}$:	a binary flag indicating if the tuple is a query tuple.
$s_{4,k}$:	SAI vector values.

Query. The query operations consist of the following three steps:

1. **First Sorting.** Sort S in the increasing order in terms of indexes (i.e. $s_{2,k}$), and then binary flags (i.e. $s_{3,k}$), as shown in Line 2 of Alg. 4. This ensures that each matrix tuple is followed by the query tuples with the same index. The resulting S is as follows:

$$\begin{pmatrix} 1: & 0 & s_{u_1^1} & \dots & s_{u_{n_1}^1} & \dots & 0 & s_{u_1^N} & \dots & s_{u_{n_N}^N} \\ 2: & 1 & 1 & \dots & 1 & \dots & N & N & \dots & N \\ 3: & 0 & 1 & \dots & 1 & \dots & 0 & 1 & \dots & 1 \\ 4: & a_1 & v_{u_1^1} & \dots & v_{u_{n_1}^1} & \dots & a_N & v_{u_1^N} & \dots & v_{u_{n_N}^N} \end{pmatrix}$$

   where query tuples from SUs $\{u_j^i\}_{j=1}^{n_i}$ share the same index as matrix tuple i, and $\sum_{i=1}^{N} n_i = N_s$ with $n_i \geq 0$ for any $1 \leq i \leq N$. After querying, it is expected that $v_{u_j^i} = a_i$ for any $1 \leq i \leq N$ and $1 \leq j \leq n_i$.
2. **Query Computing.** Perform query computing through a left-to-right traversing of the array, as shown in Line 3. By appropriate use of flags, this operation affects query tuples, leaving matrix tuples unchanged. After this operation, the entry $s_{4,k}$ (SAI value) of each matrix tuple is copied to the entries $s_{4,k}$'s of the corresponding query tuples following the matrix tuple with the same index.
3. **Second Sorting.** The tuple array S is first sorted descendingly w.r.t. binary flags (i.e. $s_{3,k}$), and then ascendingly w.r.t. IDs (i.e. $s_{1,k}$). This brings all query tuples in the first N_s positions in the array, in the order in term of their IDs.

Output. The query tuples are extracted by cutting the first N_s tuples of S.

We now show that the above algorithm is readily implementable as a circuit that takes as input $\mathcal{M}_a$ and $\{(s_i, j_i)\}_{i=1}^{N_s}$, and outputs $\{(s_i, v_i)\}_{i=1}^{N_s}$. First, Step 1 can be implemented as a circuit inputting $\mathcal{M}_a$ and $\{(s_i, j_i)\}_{i=1}^{N_s}$ and outputting the initial array S, with $\Theta(K(N+N_s))$ gates. Second, the sorting operations can be performed using a sorting network inputting the initial array and outputting the sorted array, requiring $\Theta(K(N + N_s)\log(N + N_s))$ gates. Finally, the left-to-right pass can be implemented as a circuit performing eq. (1) on each tuple, also with $\Theta(K(N + N_s))$ gates. Thus, the overall complexity of the algorithm is $\Theta(K(N + N_s)\log(N + N_s))$. When $N_s \sim \Theta(N)$, the complexity becomes $\Theta(KN \log N)$, within a logarithmic factor of the implementation in the RAM model without security guarantees.

E. Atomic Building-block Circuits

In order to implement our efficient query circuit, and thus the Batchers sorting network, we need to design the following basic building-block circuits: integer comparison, swap and multiplexing. We call them atomic building-block circuits, or atomic circuits more concisely. In our circuit design, we can make use of the "XOR-free" property of garbled circuits and aim to use as few as possible AND gates for each atomic circuit to minimize both computation and communication overhead. In the following, we denote two K-bit non-negative integers by $x = (x_K x_{K-1} ... x_2 x_1)$ and $y = (y_K y_{K-1} ... y_2 y_1)$. The atomic circuits can be designed and optimized as follows.

- **Integer comparison.** We apply directly the comparison circuit proposed in [17] for integer comparison, which is optimized by "XOR-free" property. To compare integers x and y, the circuit can be described as follows:

$$c_{i+1} = x_i \oplus (x_i \oplus c_i) \wedge (y_i \oplus c_i)$$
$$\text{subject to } 1 \leq i \leq K$$

  where for $c_1 = 0$, the comparison result $c_{K+1} = [x > y]$; for $c_1 = 1$, $c_{K+1} = [x \geq y]$. Comparison circuits for $[x < y]$ and $[x \leq y]$ can be obtained by interchanging x and y. As we can see, the K-bit integer comparison circuit only contains K AND gates.
- **Swap.** To swap x and y with a swap indicator denoted by b (If $b = 1$, swap x and y; else, remain untouched), we can use the following circuit [15].

$$x'_i = [b \wedge (x_i \oplus y_i)] \oplus x_i, \text{ and } y'_i = x'_i \oplus (x_i \oplus y_i)$$
$$\text{subject to } 1 \leq i \leq K$$

  Where

$$x' = (x'_K x'_{K-1} \cdots x'_2 x'_1) \text{ and } y' = (y'_K y'_{K-1} \cdots y'_2 y'_1)$$

  are the swapping results. This swap has been optimized with only K AND gates.
- **Multiplexing.** In the swap circuit, if only x' is returned, then we get a multiplexer $\mathtt{mux}(x, y, b)$, with $\mathtt{mux}(x, y, 0) = x$, and $\mathtt{mux}(x, y, 1) = y$.

F. Verification Mechanism

So far, we have presented a secure protocol for database-driven spectrum sharing. As we will see in next section, the protocol achieves merely security against semi-honest adversaries and requires that all participant parties follow the protocol specification honestly. However, in practice, either SDM or CSP may be malicious and deviate from the protocol arbitrarily, and thus malicious security is usually called for.

To address malicious attacks, we combine secure computations with MAC techniques to design a verification mechanism.

Our main idea is that we let the data of spectrum available information (SAI) carry some redundant information, so that their integrity can be well verified. Specifically, the SAI of each channel k in square s_{ij} is now defined as a vector of bits as follows.

$$\epsilon_k^{ij} = \sigma_k^{ij}||K_k^{ij}||MAC_L(K_k^{ij}, i||j||k||T_r||\sigma_k^{ij}) \qquad (2)$$

where some parameters are described as follows.

- σ_k^{ij} is the real SAI of channel k in square s_{ij}, with $\sigma_k^{ij} = 1$ means the channel is occupied, and $\sigma_k^{ij} = 0$ otherwise.
- K_k^{ij} is a random key of bit length L chosen by PU k for channel k in square s_{ij} during the updating at the very beginning of each round.
- T_r is the time stamp publicly designated for round r, e.g. the start date of round r.
- $MAC_L(K_k^{ij}, i||j||k||T_r||\sigma_k^{ij})$ is the first L bits of MAC using K_k^{ij} as its key and taking as input the concatenation of values i, j, k, T_r and σ_k^{ij}. The MAC algorithm and L value can be customized as needed.

Applying the verification mechanism, our secure and verifiable spectrum sharing protocol then works as follows.

- *Initialization.* All initial SAI data are prepared as eq. (2) by PUs, and then these data are shared between SDM and CSP.
- *Submission.* All PUs prepare their update request as eq. (2), while SUs prepare their query requests as before.
- *Processing.* Both SDM and CSP perform the same computations as before except that each SAI matrix entry carries more data than before.
- *Output and Verification.* Each SU receives the output in the form of eq. (2), and then the SU verifies the correctness by recomputing the MAC value and comparing it with the received MAC value.

This verification mechanism ensures that even if either SDM or CSP is malicious, it cannot update the spectrum database falsely, or manipulate the SAI data and return false query results.

V. Security Analysis

In this section, we first prove that our protocol (Protocol 2) without verification achieves security against semi-honest adversaries. We then demonstrate that our protocol with verification also defends against some common malicious attacks.

In Protocol 2, both PUs and SUs act as either input providers or output consumers, and they have no other interactions with both SDM and CSP. Therefore, Protocol 2 can be regarded as a two-party computation protocol with input from and output to the users. Intuitively, the security of the protocol implies that neither the SDM nor the CSP can learn anything about users' sensitive inputs or outputs. Formally, the definition of security against semi-honest adversaries in two-party computation can be described as follows [18].

Definition 2 (Security against semi-honest adversaries). *Let $f(x, y)$ be a function with two inputs x and y, two outputs $f_A(x, y)$ and $f_B(x, y)$. Suppose that protocol Π computes $f(x, y)$ between two parties Alice and Bob. Let $V_A^\Pi(x, y)$ ($V_B^\Pi(x, y)$, respectively) represent Alice's (Bob's) view during an execution of Π on (x, y). In other words, if $(x, \boldsymbol{r}_A^\Pi)$ (resp. $(y, \boldsymbol{r}_B^\Pi)$) denotes Alice's (Bob's) input and randomness, then*

$$V_A^\Pi(x, y) = (x, \boldsymbol{r}_A^\Pi, m_1, m_2, ..., m_t), \text{ and}$$
$$V_B^\Pi(x, y) = (y, \boldsymbol{r}_B^\Pi, m_1, m_2, ..., m_t),$$

where $\{m_i\}$ denote the messages passed between the parties. Let O_A^Π (resp. O_B^Π) denote Alice's (Bob's) output after an execution of Π on (x, y), and $O^\Pi(x, y) = (O_A^\Pi(x, y), O_B^\Pi(x, y))$. Then we say that protocol Π is secure (or preserve privacy) against semi-honest adversaries if there exist probabilistic polynomial time (PPT) simulators S_1 and S_2 such that

$$\{(S_1(x, f_A(x, y)), f(x, y))\} \overset{c}{\equiv} \{(V_A^\Pi(x, y), O^\Pi(x, y))\} \qquad (3)$$
$$\{(S_2(y, f_B(x, y)), f(x, y))\} \overset{c}{\equiv} \{(V_B^\Pi(x, y), O^\Pi(x, y))\} \qquad (4)$$

where $\overset{c}{\equiv}$ denotes computational indistinguishability.

With this formal security definition, the security of Protocol 2 without verification can be stated as Theorem 1.

Theorem 1. *As long as SDM and CSP do not collude, Protocol 2 is secure against semi-honest adversaries.*

Proof: To prove the security of Protocol 2, we first prove the security of each phase in Protocol 2 in two separate cases, depending on which party the adversary has corrupted. For each phase, we show that for all PPT adversaries, the adversary's view based on SDM and CSP's interaction is indistinguishable to the adversary's view when the corrupted party interacts with a simulator instead. In other words, we show that there exist simulators $S1$ and $S2$ that satisfy conditions (3) and (4) for each phase. Then, due to the sequential composition theory [19], we actually prove the security of the entire protocol.

(1) Secret Sharing and Submission. In this phase, each PU or SU splits its update or query request into two shares using XOR secret sharing, and sends one share to SDM and the other to CSP. SDM (resp. CSP) receives its share of all the requests, and initially holds its share of SAI matrix $\mathcal{M}$. In this process, if *SDM (resp. CSP) is corrupted*, it learns its share of all requests and $\mathcal{M}$, i.e.,

$$\{(k, [\mathcal{C}_k]_s)\}_{k=1}^K, \{[l_u]_s\}_{u=1}^{N_s}, \{[m_{ij}]_s\}_{n\times n}$$
$$(\text{resp.}\{(k, [\mathcal{C}_k]_c)\}_{k=1}^K, \{[l_u]_c\}_{u=1}^{N_s}, \{[m_{ij}]_c\}_{n\times n})$$

However, due to the security of XOR secret sharing, one share of data reveals nothing about the data, SDM (resp. CSP) in fact learns nothing about all requests and $\mathcal{M}$, except that PU k updates channel k, which is the public knowledge (i.e. public input). This implies that the view of SDM (resp. CSP) in this phase can be simulated by its public input together with some random numbers, and thus equations (3) and (4) holds. Therefore, this phase is secure according to the security definition.

(2) Update Processing. In this phase, SDM (resp. CSP) updates its share of spectrum database $\mathcal{M}$ by XORing its shares of old $\mathcal{M}$ and update requests, independently. There is no interaction between SDM and CSP. If *SDM (resp. CSP) is corrupted*, its view can be easily simulated by its shares of old $\mathcal{M}$ and update requests, and security of this phase holds.

(3) Query Processing. In this phase, based on a query circuit, CSP constructs a garbled circuit and garbles its shares of the updated $\mathcal{M}$ and query requests. CSP then sends the garbled circuit and garbled values to SDM, while holds the decoding information for output itself. Upon receiving the garbled query circuit and garbled values from CSP, SDM uses them together with its shares of updated $\mathcal{M}$ and query requests to compute the garbled query results. Finally, each SU receives its garbled query result from SDM and the corresponding decoding information from CSP, and uses them to get its plain query result. In the process, there are two corruption cases.

Case 1: CSP is corrupted. the view of CSP includes the garbled circuit and its garbled shares, i.e.,

$$G(Q), \{G([m^*_{ij}]_c)\}_{n\times n}, \{G([l_u]_c)\}_{u=1}^{N_s}$$

which can be simulated by CSP's input of this phase, i.e., CSP's shares of updated $\mathcal{M}$ and query requests ($\{[m^*_{ij}]_c\}_{n\times n}$ and $\{[l_u]_c\}_{u=1}^{N_s}$), and the public input, the query circuit, together with CSP's randomness.

Case 2: SDM is corrupted. The view of SDM includes the garbled values and the garbled circuit received from CSP, i.e.,

$$G(Q), \{G([m^*_{ij}]_c)\}_{n\times n}, \{G([l_u]_c)\}_{u=1}^{N_s}$$

and the garbled inputs and outputs of the query, i.e.,

$$\{G(m^*_{ij})\}_{n\times n}, \{G(l_u)\}_{u=1}^{N_s} \text{ and } \{G(m^*_{i_u j_u})\}_{u=1}^{N_s}$$

The garbled circuit and garbled values can be simulated by SDM's inputs of this phase, i.e., SDM's shares $\{[m^*_{ij}]_s\}_{n\times n}$ and $\{[l_u]_s\}_{u=1}^{N_s}$, the publicly known query circuit, together with a series of random numbers.

Thus, we can also find S_1 and S_2 satisfying equations (3) and (4), and the security of this phase also holds.

Since the above three phases are composed sequentially, it follows from the sequential composition theory that Protocol 2 is secure against semi-honest adversaries. $\square$

Theorem 2 states the malicious security of Protocol 2 with verification.

Theorem 2. *If either SDM or CSP is malicious, and both PUs and SUs follow the protocol honestly, Protocol 2 with verification defends against malicious attacks violating SAI data integrity, such as falsification, replacement and replay.*

Proof: In Protocol 2 with verification, SAI data are prepared with eq. (2). Due to the security in semi-honest setting, neither SDM nor CSP knows anything about the SAI data. Thus, when malicious attacks violate SAI data integrity, it is hard for the attacker (either SDM or CSP) to reconstruct SAI data with eq. (2). Actually, to reconstruct an SAI entry, the attacker needs to guess the random key (L bits) in the old one, and thus succeeds with a probability 2^{-L}. Then, if value L is properly chosen, malicious security can be ensured.

For falsification, replacement and replay attacks, we can show they all violate the SAI data integrity. In fact, a falsification attack tampers the real SAI σ_k^{ij} (c.f. eq. (2)); a replacement attack replaces an SAI entry ϵ_k^{ij} with another one of different i, j or k; and a replay attack uses an SAI entry with old time stamp. All of them require to reconstruct SAI data and thus can be defended through verification. $\square$

From security analysis above, it can be verified that the security requirements illustrated in Section II-B are satisfied. It is worth noting that, with the spectrum database and PUs' operational privacy well protected, all attacks originating from the disclose of these privacies, such as spectrum utilization based location inferring (SULI) attack [9], will be inherently prevented.

VI. Experimental Results

In this section, we conduct extensive experiments to evaluate our secure and verifiable scheme. We first demonstrate that the update processing is so fast that its running time is negligible compared to the overall running time. We then evaluate the computation and communication performance of the query operation by: (1) comparing our design with the naive design; (2) testing our design in the large-scale scenario with a large number of SUs; (3) testing our design with verification.

A. Experiment Setup

We setup the spectrum database by adopting the spectrum availability information (SAI) of Los Angles released by FCC [20], [21]. In this area, we choose 60 channels and a base station (BS) coverage region centralized in the geo-location $(-117.50°, 34.75°)$ of the scale of $100km \times 100km$, one of which with its signal coverage and service contour is shown using Google Earth in Fig. 6, and divide it into $n \times n$ squares, where n can be parameterized. We then extract the SAI by testing if each square overlaps with the service contours of these channels. We perform Monte Carlo experiments by randomly choosing different squares queried by the SUs.

We implement both our scheme and the naive query design on top of FastGC [22], a Java-based library for garbled circuits. In our experiments, we simulate spectrum database manager (SDM) and crypto-service provider (CSP) with two processes on a commodity PC, with Intel(R) Core(TM) i7 3.60GHz CPU, 16.00GB RAM, and Windows 7 OS. We focus on the following performance metrics:

- **Computation overhead**: measured by running time, the total CPU time spent by SDM and CSP.
- **Communication overhead**: measured by message volume, the total data of all messages communicated between the two parties.

B. Performance of Update

As described previously, update processing is performed independently by both SDM and CSP without any interaction. We plot the running time of SDM (or CSP) when executing our update processing with and without verification, as shown in Figs. 7(a) and 7(d). Note that, for verification, we use a key length $L = 7$ and HMAC-MD5 algorithm in this paper, which ensures that malicious attacks violating SAI data integrity could be detected with a probability more than 99%.

From the update plotting, we can see that all running times grow linearly with the number of channels (K), and dividing into less squares (i.e. smaller $N = n \times n$) improves the update

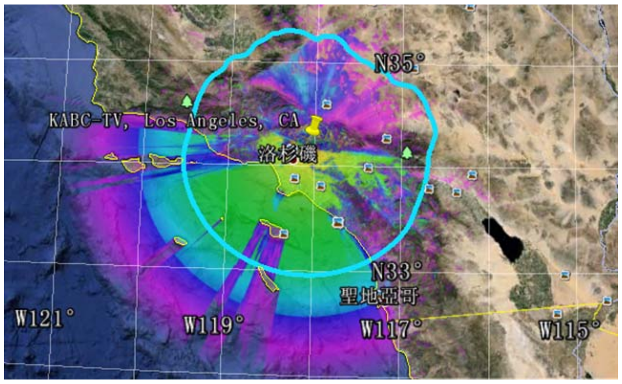


Fig. 6. The signal coverage and service contour of KABC-TV located at (−118.067020°, 34.226950°), whose channel is ch7 and ERP is 28.700kW.

performance. Most importantly, all running times range on the order of seconds due to the absence of cryptographical operations, and thus are negligible compared to the overall running times of our scheme ranging on the order of minutes, as we will see below.

C. *Performance of Query*

1) Our design vs Naive design: We carry out a simulation to compare the query performance of our design and the naive design with $N = 100 \times 100$. Query requests are randomly generated over the squares. Since the naive query is computationally-intensive, we let N_s varies from 50 to 300. The experimental results are shown in Fig. 7(b) and 7(e).

Figs. 7(b) and 7(e) demonstrate that: (1) our design is more efficient than the naive design in term of computation and communication performance when N_s exceeds 100, and the performance gaps become larger as N_s increases; (2) both running times and message volumes of our design grow much more slowly than those of the naive design.

The reason for (1) is that our design gains efficiency by leveraging parallelism of multiple query operations, while the computation and communication overheads of the naive design scale linearly to the number of query operations. The more query operations are executed in parallel, the more efficiency is gained by our design over the naive design. The reason for (2) is that when the number of query operations is much smaller compared to the number of squares (i.e., $N = n \times n$), our design takes nearly constant running times and message volumes since the number of tuples being sorted is nearly the same, while the naive design takes running times and message volumes linear to the number of query operations due to the linear growth of computation and communication overheads.

2) Large-scale system performance: In large-scale systems, the number of SUs N_s may be comparable to the total number of squares N. We evaluate our query performance in this context with $N = 100 \times 100$, by varying N_s from 1000 to 10000 for $K = 20$, 40 and 60, respectively.

Figs. 7(c) and 7(f) trace the experimental results. We can see that the running times and message volumes grow with N_s almost linearly. They also grow with K in a roughly linear fashion. The running times spent are within 2 hours, and the message volumes are within 6GB, which is a acceptable for spectrum sharing on a weekly basis.

TABLE II. PERFORMANCE OF QUERY WITH VERIFICATION

N_s	50	100	150	200	250	300
Time (min, $K = 20$)	22.59	22.61	23.45	24.12	24.77	25.67
Time (min, $K = 40$)	42.15	43.26	45.30	46.30	47.80	49.07
Data (MB, $K = 20$)	1136	1174	1212	1251	1289	1330
Data (MB, $K = 40$)	2149	2219	2291	2363	2434	2509

3) Query with verification: To test query with verification, we apply the same configure as Sec. VI-B. We then use $N = 50 \times 50$, vary N_s from 50 to 300, with $K = 20$ and 40, respectively. Tab. II lists the running times and message volumes. It can be seen that both performance metrics grow slowly with N_s due to the complexity $(N + N_s)\log(N + N_s)$ of our query algorithm, and their values are acceptable for practical applications.

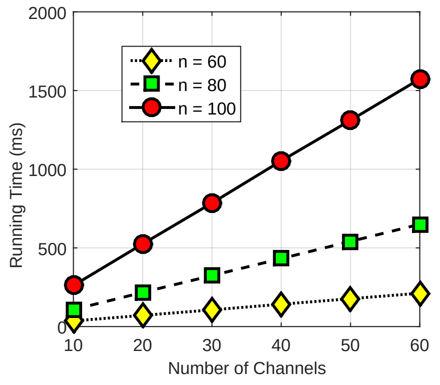
VII. Related Work

In this section, we summarize existing work related to ours.

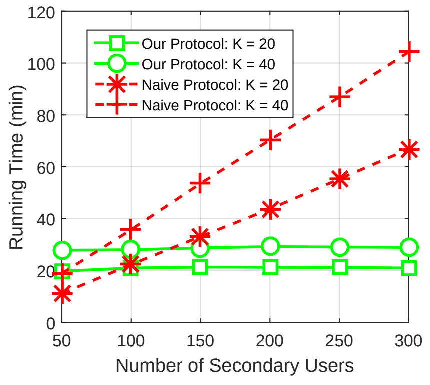
Privacy preservation in location-based service (LBS). Existing work on privacy preservation in LBS typically uses the K-anonymity technique [5][6], collaborative privacy protection such as mix-zone [7], and differential privacy such as geo-indistinguishability [8]. However, K-anonymous location privacy protection normally requires a trusted server, collaborative privacy protection suffers extra cost caused by collaboration among users, while differential privacy protection requires querying with users' obfuscated locations and thus getting probabilistically inaccurate query results, which make them unadaptable in protecting SUs' location privacy. Moreover, these methods cannot be applied to protect the PUs' operational privacy during the spectrum database update.

Security in cognitive radio networks (CRNs). While being a viable option to improve spectrum utilization, CRNs present many specifical security vulnerabilities [3]. Attacks against sensing-driven spectrum sharing have been identified in [23], including primary user emulation (PUE) attack [24], spectrum sensing data falsification (SSDF) attack [25], etc. For database-driven spectrum sharing, security threats concern the privacy of both primary users (PUs) and secondary users (SUs), and the database access [3]. To address these security threats, several solutions have been proposed to prevent rogue transmissions (e.g. rule-based [26] and ontology-based policies [27]) or to punish non-compliant transmitters (e.g. localization [28] and punishment [29]). However, very limited work has been done on privacy preservation in CRNs.

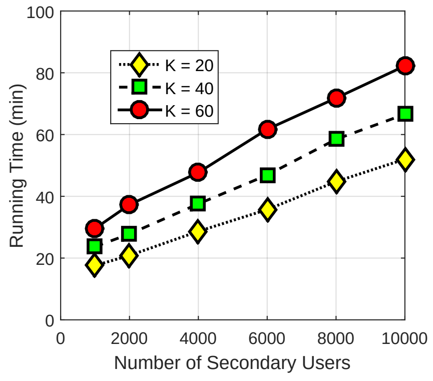
Privacy preservation in database-driven CRNs. Recently, the authors of [9] proposed PriSpectrum, a scheme that protects SUs' location privacy in database-driven CRNs. The authors of [30] developed a location privacy-preserving spectrum auction that can be applied in database-driven CRNs. However, existing work has considered neither PUs' operational privacy, nor correctness verification of spectrum database operations. Different from [9], [30], which focused on solely protecting SUs' location privacy in database-driven CRNs, our work not only enables both secure update and query



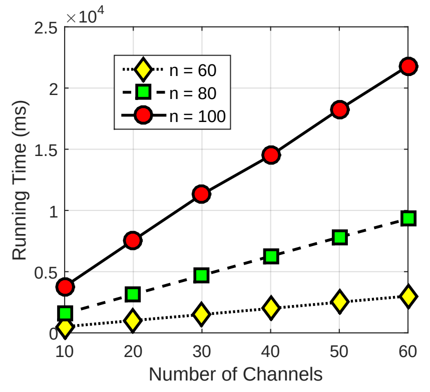
(a) Update without verification



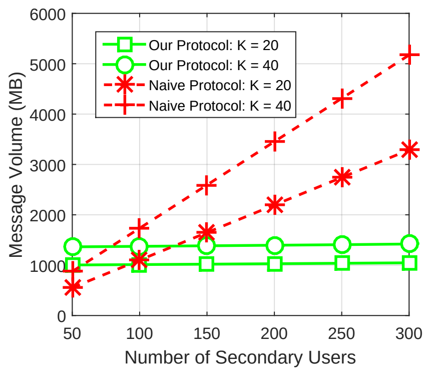
(b) Time: naive vs our.



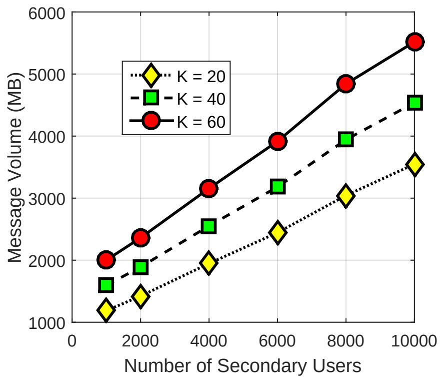
(c) Running time



(d) Update with verification



(e) Message: naive vs our.



(f) Message volume

Fig. 7. Performance evaluation. 7(a) and 7(d) show the running times of the update processing for SDM (or CSP) with or without verification. 7(b) and 7(e) show the query performance comparison of our design and the naive design in both computation and communication overheads. 7(c) and 7(f) show the query performance of our design in large-scale systems in both computation and communication overheads.

operations, protecting both PUs' operational privacy and SUs' location privacy, and the spectrum database, but also enables correctness verification of database operations in the presence of malicious attacks.

Two-party computation for privacy preservation. There is a research strand that introduces a third party and applied the two-party computation techniques for privacy preservation, such as [31][32][15]. Our work follows this direction. To our knowledge, we are the first to apply two-party computation techniques (e.g. garbled circuits and cryptographical security formulation) to privacy preservation in the context of database-driven CRNs.

VIII. Conclusion

In this paper, we have proposed a secure and verifiable scheme for database-driven spectrum sharing, allowing both update and query operations. Our scheme not only achieves cryptographical security against semi-honest adversaries, but also defends against some common malicious attacks. Specifically, it reveals nothing about the primary users' (PUs') operational information (except the public knowledge transmitter IDs), the secondary users' (SUs') location information and the spectrum database to SDM, CSP and all users, resulting that each SU gets its respective query result. Also, the correctness of each SU's query result can be well verified, so that any malicious attack violating the integrity of spectrum availability information could be detected. To our knowledge, this is the first secure and verifiable database-driven spectrum sharing scheme protecting both PUs' operational privacy and SUs' location privacy.

Acknowledgment

The work of Zhili Chen is supported by the Natural Science Foundation of China under Grant Nos. 61572031 & 61202407; The work of Hong Zhong is supported by the Natural Science Foundation of China under Grant No. 61572001 and the Natural Science Foundation of Anhui Province under Grant No. 201508085QF132.

References

[1] D. Gurney, G. Buchwald, L. Ecklund, S. Kuffner, and J. Grosspietsch. Geo-location database techniques for incumbent protection in the TV white space. In *Proc. IEEE DySPAN*, pages 1–9, 2008.

[2] Federal Communications Commission. Third memorandum opinion and order. http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-12-36A1.pdf, 2012.

[3] J.M. Park, J.H. Reed, A.A. Beex, and T.C. Clancy. Security and enforcement in spectrum sharing. *Proceedings of the IEEE*, 102(3):270–281, 2014.

[4] Federal Communications Commission. Amendment of the commissions rules with regard to commercial operations in the 3550-3650 mhz band. In *Notice of Proposed Rulemaking and Order*, 2012.

[5] T. Xu and Y. Cai. Feeling-based location privacy protection for location-based services. In *Proc. of ACM CCS*, pages 348–357, 2009.

[6] K. Vu, R. Zheng, and J. Gao. Efficient algorithms for k-anonymous location privacy in participatory sensing. In *Proc. of IEEE INFOCOM*, pages 2399–2407, 2012.

[7] J. Freudiger, M. Manshaei, J. Hubaux, and D. Parkes. On noncooperative location privacy: A game-theoretic analysis. In *Proc. of ACM CCS*, pages 324–337, 2009.

[8] M.E. Andrs, N.E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proc. of ACM CCS*, 2013.

[9] Z. Gao, H. Zhu, Y. Liu, M. Li, and Z. Cao. Location privacy in database-driven cognitive radio networks: Attacks and countermeasures. In *Proc. of IEEE INFOCOM*, pages 2751–2759, 2013.

[10] A. C.-C. Yao. How to generate and exchange secrets. In *Proc. FOCS*, 1986.

[11] Y. Lindell and B. Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2), 2009.

[12] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi. Secure and efficient outsourcing of sequence comparisons. In *European Symposium on Research in Computer Security*, pages 505–522. Springer, 2012.

[13] Z. Chen, L. Huang, and L. Chen. ITSEC: An information-theoretically secure framework for truthful spectrum auctions. In *Proc. of INFOCOM*, pages 2065–2073, 2015.

[14] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conference*, 1968.

[15] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *Proc. ACM CCS*, 2013.

[16] O. Goldreich. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 1996.

[17] V. Kolesnikov, A. R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Proc. CANS*, 2009.

[18] O. Goldreich. *Foundations of Cryptography: Volume 2-Basic Applications*. Cambridge University Press, 2004.

[19] C. Hazay and Y. Lindell. *Efficient secure two-party protocols: Techniques and constructions*. Springer, 2010.

[20] TV fool. Available:http://www.tvfool.com/, 2015.

[21] Federal Communications Commission. TV query broadcast station search. Available:https://www.fcc.gov/encyclopedia/tv-query-broadcast-station-search, 2015.

[22] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proc. USENIX Security*, 2011.

[23] K. Bian and J.M. Park. MAC-layer misbehaviors in multi-hop cognitive radio networks. In *Proc. of US-Korea Conference on Science, Technology, and Entrepreneurship*, 2006.

[24] R. Chen, J.M. Park, and J. Reed. Defense against primary user emulation attacks in cognitive radio networks. *IEEE J. Sel. Areas Commun.*, 26(1):25–37, 2008.

[25] R. Chen, J.M. Park, and K. Bian. Robust distributed spectrum sensing in cognitive radio networks. In *Proc. of IEEE INFOCOM*, pages 1876–1884, 2008.

[26] B. Bahrak, A. Deshpande, M. Whitaker, and J.M. Park. Bresap: A policy reasoner for processing spectrum access policies represented by binary decision diagrams. In *Proc. of IEEE DySPAN*, 2010.

[27] M. Kokar and L. Lechowicz. Language issues for cognitive radio. *Proc. IEEE*, 2009.

[28] T. He, C. Huang, B.M. Blum, J.A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Proc. of ACM MobiCom*, pages 81–95, 2003.

[29] K. Woyach, A. Sahai, G. Atia, and V. Saligrama. Crime and punishment for cognitive radios. In *Proc. of 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 236–243, 2008.

[30] S. Liu, H. Zhu, R. Du, C. Chen, and X Guan. Location privacy preserving dynamic spectrum auction in cognitive radio network. In *Proc. of ICDCS*, pages 256–265, 2013.

[31] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. EC*, 1999.

[32] Q. Huang, Y. Tao, and F. Wu. SPRING: A strategy-proof and privacy preserving spectrum auction mechanism. In *Proc. INFOCOM*, 2013.