TrustSAS: A Trustworthy Spectrum Access System for the 3.5 GHz CBRS Band

Mohamed Grissa⋆, Attila A. Yavuz‡, and Bechir Hamdaoui⋆
⋆ Oregon State University, grissam,hamdaoui@oregonstate.edu
‡ University of South Florida, attilaayavuz@usf.edu

***Abstract*—As part of its ongoing efforts to meet the increased spectrum demand, the Federal Communications Commission (FCC) has recently opened up 150 MHz in the 3.5 GHz band for shared wireless broadband use. Access and operations in this band, aka Citizens Broadband Radio Service (CBRS), will be managed by a dynamic spectrum access system (SAS) to enable seamless spectrum sharing between secondary users (SUs) and incumbent users. Despite its benefits, SAS's design requirements, as set by FCC, present privacy risks to SUs, merely because SUs are required to share sensitive operational information (e.g., location, identity, spectrum usage) with SAS to be able to learn about spectrum availability in their vicinity. In this paper, we propose $TrustSAS$, a trustworthy framework for SAS that synergizes state-of-the-art cryptographic techniques with blockchain technology in an innovative way to address these privacy issues while complying with FCC's regulatory design requirements. We analyze the security of our framework and evaluate its performance through analysis, simulation and experimentation. We show that $TrustSAS$ can offer high security guarantees with reasonable overhead, making it an ideal solution for addressing SUs' privacy issues in an operational SAS environment.**

***Index Terms*—Spectrum access system, Citizens Broadband Radio Service, spectrum databases, Blockchain, privacy.**

I. Introduction

THe Federal Communications Commission (FCC) continues its effort towards promoting dynamic access to spectrum resources, and has recently promulgated the creation of the Citizens Broadband Radio Service (CBRS) in the 3.5 GHz band (3550 - 3700 MHz) [1]. This opens up previously protected spectrum, used by the US Navy and other DoD members, for dynamic and opportunistic spectrum sharing. In its CBRS report [1], [2], FCC prescribes the use of a centralized spectrum access system (SAS) to govern CBRS sharing among incumbent and secondary users. Like the case of TV white space (TVWS) access, SAS comprises multiple geolocation spectrum databases (DBs), operated (typically) by different administrators and are required to communicate amongst themselves to ensure frequency use information consistency. Also, like in TVWS, SUs need to query the DBs using their exact location information to be able to learn about CBRS spectrum opportunities in their vicinity.

SAS supports three types of users: primary users (PUs), priority access license (PAL) users, and general authorized access (GAA) users. PUs are top/first tier users with the highest priority, while new CBRS users, considered as secondary users, operate either at the second tier as PAL users or at the third tier as GAA users [3]. PAL users are assigned through competitive auction and have priority over GAA users, but they are required to vacate the spectrum upon the return of PUs. GAA users, on the other hand, operate opportunistically, in that they need to query the DBs to learn about which spectrum portions are available—not being used by higher tier (PU or PAL) users. Even though both PAL and GAA users are considered as secondary users, in the remaining parts of this paper, for ease of illustration, SU refers to a GAA user, since only GAA users need to query DBs to learn spectrum availability; PAL users acquire spectrum access via bidding.

A. Key SAS Requirements

As stipulated by FCC [1], SAS's capabilities will exceed those of TVWS [4], allowing a more dynamic, responsive and generally capable support of a diverse set of operational scenarios and heterogeneous networks [5]. While some of FCC's design requirements for SAS, such as the ability to authenticate users, hold users accountable for rule and policy violation, and to protect against unauthorized database access and tampering, are similar to TVWS systems, other requirements are only specific to SAS, which include [2]:

- **Information gathering and retention**: SUs must keep SAS informed about their current operating parameters and channel usage information at all time, so that SAS can maintain accurate and up-to-date frequency usage information. While this is mandatory in SAS, it is only optional in TVWS.
- **Coexistence**: SAS is required to coordinate the interactions among PAL and GAA users to ensure interference-free coexistence among all CBRS users [2], [6]. This is different from TVWS systems, which focus primarily on protecting PUs, and not on ensuring coexistence among SUs.
- **Auditability**: SAS must maintain audit logs of all system operations [7], including DB write operations, user membership status changes, etc. SAS uses these logs to verify and ensure compliance with regulatory rules and policies.

It is then important that these requirements be met when designing SAS. The challenge, however, is that meeting them gives rise to some serious privacy issues, thereby impacting the adoption of this promising technology.

B. Privacy Issues in SAS

A subtle privacy concern arises in SAS, which pertains merely to the fact that SUs are required to share sensitive operational information with DBs in order for them to obtain spectrum availability information [2]. This information, which may include SUs' sensitive data, such as their locations, identities, spectrum usage, and transmission parameters, may be collected by an adversary or a malicious SAS administrator and be exploited for economic, political, or other purposes [8]. For instance, fine-grained location information can easily reveal other personal information about SUs including their behavior, health condition, personal habits or beliefs [9].

It may not be acceptable for most users to expose such a sensitive information, especially in the presence of malicious entities that can exploit it for malicious purposes [9]–[11]. Such privacy risks may hinder the wide adoption of this promising spectrum sharing technology. Calls are starting to arise within the wireless community to raise awareness about this issue as it is the case with Federated Wireless in their comments to FCC regarding its report and order [2]. Therefore, it is necessary to design mechanisms that can protect SUs' sensitive information while at the same time abiding by FCC's rules and policies prescribed for SAS.

C. Contributions and Paper Organization

Most of SAS' rules require SUs to share a great deal of their sensitive information, which conflict with SUs' privacy objectives. As a result, we are facing a dilemma: On one hand, all SAS entities need to comply with SAS's requirements to have a stable, interference-free radio environment. On the other hand, it is important to offer privacy guarantees to SUs so as to promote this new spectrum sharing technology. This dilemma makes the task of designing SAS mechanisms that provide privacy guarantees while complying with SAS's requirements and rules very challenging.

We strongly envision that the public's (long-term) acceptance of the SAS paradigm will greatly depend on the robustness and trustworthiness of SAS vis-a-vis of its ability to address these privacy concerns. Therefore, in this work, we propose $TrustSAS$, a trustworthy SAS design framework that aims to achieve these two conflicting goals. More specifically, $TrustSAS$ combines and synergizes state-of-the art cryptographic techniques with blockchain technology in an innovative way to address these privacy issues while complying with FCC's regulatory design requirements. To the best of our knowledge, this work is the first to address such issues within the context of SAS and CBRS.

We first provide in Section II a high-level overview of our framework to help grasp the big picture. Then, in Section III, we provide a detailed description of the framework. The security analysis and performance evaluation are provided in Sections IV and V, and the paper is concluded in Section VI.

II. System and Framework Overview

In this section, we present the system architecture and provide a high-level overview of $TrustSAS$. Fig. 1 can be referred to throughout this section to facilitate the description.

A. Architectural Components

As illustrated in Fig. 1, $TrustSAS$ comprises three main architectural entities: FCC, multiple DBs, and multiple SUs. Without loss of generality, throughout the paper, we use FCC to refer to FCC itself, or to any trusted third-party entity that is appointed by FCC to act on its behalf. In $TrustSAS$, FCC is responsible for enforcing compliance with regulatory requirements, providing system keying materials, handling the registration of SUs, and granting them permissions to join

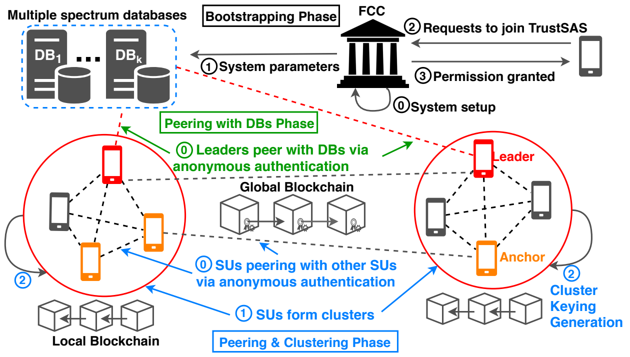

Multiple spectrum databases
Bootstrapping Phase
FCC
DB1 ... DBk
② Requests to join TrustSAS
① System parameters
③ Permission granted
⓪ System setup
Peering with DBs Phase
⓪ Leaders peer with DBs via anonymous authentication
Leader
Global Blockchain
Anchor
⓪ SUs peering with other SUs via anonymous authentication
②
② Cluster Keying Generation
① SUs form clusters
Local Blockchain
Peering & Clustering Phase

Fig. 1. TrustSAS Architecture and Initial Operations

$TrustSAS$. $TrustSAS$ leverages and relies on the existence of multiple DBs for spectrum access, each typically run by a different administrator. These DBs are assumed to be synchronized and to be sharing the same content, as mandated by FCC. Also, $TrustSAS$ supports multiple SUs, including a set of pre-registered SUs to be deployed specifically for playing the role of anchor nodes. These anchor SUs serve to establish a backbone peer-to-peer (p2p) network that can be discoverable and joinable by new SUs.

The content of each DB can be viewed/modelled as an $r \times b$ matrix $\mathcal{D}$ of size η bits, where r is the number of records in the database, each of size b bits. Each record in $\mathcal{D}$ is a unique combination of a cell number, representing the location, a channel number, and other transmission parameters (e.g., max transmit power, duration, etc). In $TrustSAS$, each record in $\mathcal{D}$ contains a smart contract that is to be created by DBs to define channel usage rules, such as the maximum number of SUs allowed to transmit simultaneously in a given location, SU's maximum transmit power, etc. With these smart contracts, $TrustSAS$ ensures fair sharing of the spectrum resources, and limits the interference among SUs, thus satisfying the *coexistence* requirement, stated in Section I-A. For simplicity, we assume that channel usage is permitted over a fixed duration independently from the channel, and that SUs need to query DBs for an updated channel availability information periodically every $\mathcal{T}_{epoch}$, where $\mathcal{T}_{epoch}$ is a tunable system design parameter. The geographical area serviced by $TrustSAS$ is modeled as a grid of $N \times N$ cells of equal sizes, and an SU's location is expressed through the grid's cell index.

B. TrustSAS Initial Setup

The first phase needed for setting up $TrustSAS$ is bootstrapping (see Fig. 1), during which FCC creates the system parameters and keys, specific to $TrustSAS$, and shares them with DBs. Also, SUs first need to register and request SAS access privileges from FCC before they can join $TrustSAS$. Once registered, FCC provides the joining SU with the anchor SU list, membership keys, and the procedure necessary for the SU to authenticate with and join $TrustSAS$. Note that, in $TrustSAS$, all messages communicated between the SUs and the DBs are established over secure channels, so as to ensure that the spectrum queries are authenticated, private, and not tampered with. Secure channels will be established via traditional mechanisms, and such mechanisms are ignored in this framework to keep the focus on the other security aspects. This phase is detailed in Section III-A1.

The second setup phase consists of establishing the underlying network infrastructure. Registered SUs that join $TrustSAS$ will maintain communication with one another via an overlay p2p network, and a newly joining SU will rely on anchor SUs to discover and join the p2p network. $TrustSAS$ relies on an anonymous digital signature technique, explained in Section III-A, to enable all these SUs to anonymously authenticate and verify each other's legitimacy when peering with one another. This anonymous authentication will also enable SUs to enjoy system services anonymously, yet in a verifiable way, to break the link between their sensitive operational data and their true identities.

$TrustSAS$ adopts a clustering approach, where joined SUs group themselves into clusters and elect cluster leaders, with the leaders being responsible for representing their SUs for interacting with other system entities. Not only will this improve $TrustSAS$ scalability, but also protect SUs' privacy, as it will limit the interaction with DBs to only cluster leaders. Once clusters are established, SUs within each cluster distributively and collaboratively generate their cluster-specific keys, which will be used later for blockchain related operations inside the cluster and for signing cluster-wise spectrum agreements. This phase is detailed in Section III-A2.

Once clusters are formed, the last setup phase is for the leaders to anonymously authenticate with DBs, and upon successful authentication, these DBs will join and be part of the established p2p network. This way, DBs will not be involved in the initial clustering of SUs, and therefore they will not be able to infer the SUs' location information. This phase is detailed in Section III-A3.

C. TrustSAS Main Operations

1) Querying Spectrum Availability Information: Each cluster leader acts on behalf of its SU members and privately queries DBs for spectrum availability information. Even though the true identities of all SUs, including leaders, are hidden in $TrustSAS$, this is not sufficient to preserve their operational privacy. In fact, since each record in DBs is associated with a unique location, DBs may infer the location of the leaders from their queries and can still use this information for tracking purposes. To prevent this, $TrustSAS$ protects the leaders' queries through the adoption of multi-server private information retrieval (PIR) protocol [12], which enables a user to retrieve a record from multiple databases while preventing the databases from learning any information about the record or the user requesting it. After learning the spectrum availability information, members of each cluster will distributively reach an agreement on how the spectrum resources are to be shared among them. Detailed description of this operation is provided in Section III-C.

2) Notifying about Spectrum Usage: Once a spectrum assignment agreement is reached, the cluster leader will notify the DBs about the spectrum portions used by its cluster members, as well as about other information, such as aggregate transmit power on each used channel, duration of channel use, etc., as required by FCC. $TrustSAS$ uses this information to build knowledge of the spectral environment and to maintain an accurate availability information to comply with the information gathering and retention requirement. As we discuss in more details in Section III, $TrustSAS$ ensures that cluster leaders report an accurate and non-altered spectrum usage information that is easily verifiable. Other leaders and DBs will distributively reach an agreement about the validity of this information, which, if valid, will be updated to DBs' records. Detailed description of this is provided in Section III-D.

III. The Proposed Framework: $TrustSAS$

$TrustSAS$ relies on permissioned blockchains [13] to keep track of system and cluster activities. Blockchains are also used as a platform to handle agreements between entities at both the cluster and system levels. This is achieved thanks to permissioned blockchains' underlying *Byzantine fault tolerant* (BFT) consensus mechanism [13], which enables participants to reach agreements on block updates even when Byzantine nodes are present. Throughout the description of $TrustSAS$, before an entity submits and adds a block to a blockchain, we assume that the block is first signed by the entity and then validated via BFT by the validators of the blockchain. We now describe the different algorithmic components of $TrustSAS$.

A. System Setup

The first component of $TrustSAS$, depicted in Alg. 1, consists of setting up the system parameters and the required keys at initialization, which is done in three phases.

1) Bootstrapping Phase (Alg. 1, steps 2-10): $TrustSAS$ ensures that SUs activities are anonymous, yet verifiable, by leveraging Intel's anonymous digital signature, known as enhanced privacy ID (EPID) [14]. EPID allows any SU to prove its membership legitimacy to other $TrustSAS$ entities, without revealing its true identity, using zero-knowledge proof [15]. EPID also enables access revocation of misbehaving SUs anonymously, by maintaining and using a revocation list $\mathcal{L}$ based on SUs' signatures. EPID typically runs four procedures. The first, EPID.Setup, is run by the FCC as the first step of the Bootstrapping phase (step 2, Alg. 1) and outputs two system keys: *Membership Verification Public Key* ($\mathcal{K}_{pk}$) and *Membership Issuing Secret Key* ($\mathcal{K}_{sk}$). The first key, $\mathcal{K}_{pk}$, is shared among all entities of $TrustSAS$, and used by SUs and DBs to anonymously verify the membership legitimacy of another SU. The second key, $\mathcal{K}_{sk}$, is kept secret and used only by FCC to create a unique *Membership Private Key*, sk_{SU}, for each joining SU, a key that the SU uses to prove its membership legitimacy to the other system members anonymously. *We iterate again that FCC will be used throughout to refer to either FCC itself or any third-party entity that is appointed by FCC to govern on its behalf.*

The second procedure, EPID.Join, is run interactively between each joining SU and FCC, and takes as input $\mathcal{K}_{pk}$ and FCC's public key $\mathcal{K}_{FCC}$, as illustrated in steps 4 and 9 of

Alg. 1. It results in SU obtaining $\mathcal{K}_{pk}$ and sk_{SU}. The third procedure, EPID.SIGN, allows an SU to anonymously prove its membership legitimacy and that it does not belong to the revocation list (i.e., its signature over a challenge message, m, does not belong to $\mathcal{L}$). Note that EPID signatures produced by the same SU are linkable; this prevents any malicious SU from forging multiple signatures on behalf of other SUs. To validate the EPID signature of joining SU, a verifier uses the fourth procedure, EPID.VERIFY, using the membership public key, $\mathcal{K}_{pk}$, by checking that SU's signature is not in $\mathcal{L}$.

Algorithm 1 *TrustSAS* setup

```
1:  function TWOWAYEPID($A, B, \mathcal{K}_{pk}, \mathcal{L}$)
        User $A$ sends a challenge $m_A$ to user $B$
        User $B$ sends a challenge $m_B$ to user $A$
        $A$: $(\Sigma_A, \mathcal{P}_A) \leftarrow$EPID.SIGN($sk_A, \mathcal{K}_{pk}, m_B, \mathcal{L}$ )
        $B$: $v_A \leftarrow$EPID.VERIFY($\mathcal{K}_{pk}, m_B, \Sigma_A, \mathcal{P}_A, \mathcal{L}$ )
        $B$: $(\Sigma_B, \mathcal{P}_B) \leftarrow$EPID.SIGN($sk_B, \mathcal{K}_{pk}, m_A, \mathcal{L}$ )
        $A$: $v_B \leftarrow$EPID.VERIFY($\mathcal{K}_{pk}, m_A, \Sigma_B, \mathcal{P}_B, \mathcal{L}$ )
        return $v_A \wedge v_B$
    Bootstrapping phase
2:  FCC: $(\mathcal{K}_{pk}, \mathcal{K}_{sk}) \leftarrow$ EPID.SETUP($\kappa$) ▷ $\kappa$: security level
3:  FCC shares $\mathcal{K}_{pk}$ with $DB$s
4:  $(sk_{SU}, \mathcal{K}_{pk}) \leftarrow$EPID.JOIN($\mathcal{K}_{pk}, \mathcal{K}_{FCC}$) $\forall SU \in \mathcal{A}$
5:  for $SU\ k \in \mathcal{A}$ do
6:      for $SU\ l \in \mathcal{A} \setminus \{k\}$ do
7:          TWOWAYEPID($k, l$)
8:  All $SU$s $\in \mathcal{A}$ peer up with each other
9:  Joining $SU$: $(sk_{SU}, \mathcal{K}_{pk}) \leftarrow$ EPID.JOIN($\mathcal{K}_{pk}, \mathcal{K}_{FCC}$)
10: FCC shares $\mathcal{A}$ with joining $SU$
    Peering and clustering phase
11: $SU$ joins and discovers the p2p network through $\mathcal{A}$
12: $SU$ runs TWOWAYEPID() with each peer
13: $SU$s of the overlay network form clusters $\{\mathcal{C}^{(i)}\}_{1 \le i \le n_{\mathcal{C}}}$
14: $SU$s $\in \mathcal{C}^{(i)}$ elect a leader $SU_L^{(i)}$, $\forall\ 1 \le i \le n_{\mathcal{C}}$
15: $SU$s $\in \mathcal{C}^{(i)}$ maintain a local blockchain $\mathcal{BC}^{(i)}$
16: $SU$s $\in \mathcal{C}^{(i)}$ run steps 2-6 of REKEYING($\mathcal{C}^{(i)}$) (Alg. 2)
    Peering with $DB$s
17: $DB$s form validators set $\mathcal{V}$
18: Global blockchain $\mathcal{BC}$ is created with validators $\in \mathcal{V}$
19: $DB$s $\in \mathcal{V}$ and FCC maintain full copy of $\mathcal{BC}$
20: for $i = 1, \cdots, n_{\mathcal{C}}$ do
21:     $SU_L^{(i)}$ authenticates with $DB$s using EPID
22:     $SU_L^{(i)}$ peers up with $DB$s and becomes a validator
23:     $SU_L^{(i)}$ submits $y^{(i)}$ to $\mathcal{BC}$
24:     $SU_L^{(i)}$ requests a beacon $\beta^{(i)}$ from a $DB$
25:     $DB$ sends an EPID challenge $m$ to $SU_L^{(i)}$
26:     $SU_L^{(i)}$:$(\Sigma_L, \mathcal{P}_L) \leftarrow$EPID.SIGN($sk_L, \mathcal{K}_{pk}, m, \mathcal{L}$)
27:     $DB$ verifies $(\Sigma_L, \mathcal{P}_L)$ with EPID.VERIFY()
28:     $DB$ issues $\beta^{(i)}$ to $SU_L^{(i)}$ and submits it to $\mathcal{BC}$
29:     $SU_L^{(i)}$ selects $SU$s $\in \mathcal{C}^{(i)}$ into $\mathcal{R}^{(i)}$
30:     Every $\mathcal{T}_\beta$, $SU$s $\in \mathcal{R}^{(i)}$ transmit $\beta_i$ for a duration $d$
```

TrustSAS also requires that some SUs be appointed to serve as anchor nodes. These SUs need to run the TWOWAYEPID subroutine (Alg. 1, step 1) among themselves to authenticate each other anonymously before they peer up and initiate the overlay p2p network. Later on, every joining SU, that obtained its sk_{SU} through EPID.JOIN, will also get the list of anchor nodes, denoted by $\mathcal{A}$ throughout, from FCC.

2) Joining and Clustering Phase (Alg. 1, steps 11-16): Every joining SU uses the list $\mathcal{A}$ to discover and join the on-going p2p network. The joining SU then needs to authenticate with its peers and verify their legitimacy via TWOWAYEPID (Alg. 1, step 1). After enough SUs have joined *TrustSAS*, these SUs will form clusters based on their locations; this may require the SUs to expose their locations to other SUs, but it should be no issue at this point since DBs are not part of the p2p network yet. The members of each $\mathcal{C}^{(i)}$ will also maintain a cluster (local) blockchain, $\mathcal{BC}^{(i)}$, to log and keep track of key events taking place in the cluster.

TrustSAS requires SUs of each cluster to serve as witnesses with respect to any cluster-related statement that is shared by the leader with the system. This is to prevent the leader from maliciously reporting incorrect information that was not validated by members of the cluster. To ensure this, *TrustSAS* adopts the robust (t, n)-threshold BLS (TBLS) signature scheme [16]. TBLS requires no more than (any) $t_i + 1$ of the n_i SUs in $\mathcal{C}^{(i)}$ to collaboratively create a cluster signature over a statement. For this, members of each $\mathcal{C}^{(i)}$ will have to run the REKEYING operation described in Alg 2, among the n_i SUs of $\mathcal{C}^{(i)}$, to jointly generate the keys required for performing such distributed (t_i, n_i)-TBLS signatures within $\mathcal{C}^{(i)}$. This is achieved by running TBLS's distributed key generation (DKG) [17] operation which will result in each SU j in $\mathcal{C}^{(i)}$ obtaining three keys: Cluster Public Key, $y^{(i)}$, which is shared among all SUs in $\mathcal{C}^{(i)}$, Cluster User Secret Key, $x_j^{(i)}$, and Cluster User Public Key, $z_j^{(i)} = g^{x_j^{(i)}}$. The Cluster User Secret Keys $(x_1^{(i)}, \cdots, x_{n_i}^{(i)})$ are a (t_i, n_i)-threshold secret sharing of the private key $x^{(i)} = \log_g y^{(i)}$. These shares are constructed using Shamir secret sharing [18] such that any subset of $t_i + 1$ SUs from $\mathcal{C}^{(i)}$ can recover $x^{(i)}$ using Lagrange interpolation. Cluster User Public Keys represent SUs' pseudonyms within $\mathcal{C}^{(i)}$ and are also used to identify SUs' transactions in the local blockchain, $\mathcal{BC}^{(i)}$. In addition to DKG, TBLS comprises four other operations:

- SIGNSHAREGEN: It enables each SU j to compute the signature share $\sigma_j^{(i)}$ over a message m to be signed by $\mathcal{C}^{(i)}$.
- SIGNSHAREVERIF: It enables members of $\mathcal{C}^{(i)}$ to verify SU j's signature share $\sigma_j^{(i)}$ against its public key $z_j^{(i)}$.
- SIGNRECONSTRUCT: The leader of a cluster collects a set of $t_i + 1$ signature shares of a message m, $\mathcal{H}_i$, verified using SIGNSHAREVERIF, from $t_i + 1$ SUs. It combines these shares using Lagrange interpolation, via the Lagrange coefficients that were calculated in DKG, and reconstructs the complete cluster signature.
- GROUPSIGNVERIF: Used to verify the cluster-generated

signature against $\mathcal{C}^{(i)}$'s public key $y^{(i)}$.

Note that TBLS does not require reconstructing $x^{(i)}$ during the signing process. Even after repeated signing, no SU could learn any information about $x^{(i)}$ that would enable it to create signatures without t_i other SUs [19]. We refer the reader to [16] for more details about TBLS.

Algorithm 2 Rekeying within $\mathcal{C}^{(i)}$

```
1: procedure REKEYING($\mathcal{C}^{(i)}$)
2:     $\{y^{(i)}, x_1^{(i)}, \cdots, x_{n_i}^{(i)}, z_1^{(i)}, \cdots, z_{n_i}^{(i)}\} \leftarrow$TBLS.DKG($I$)
3:     for $SU\ j \in \mathcal{C}^{(i)}$ do
4:         $(\Sigma_j, \mathcal{P}_j) \leftarrow$ EPID.SIGN($sk_j, \mathcal{K}_{pk}, z_j^{(i)}, \mathcal{L}$ )
5:         $\varrho_j \leftarrow$ TBLS.SIGNSHAREGEN($x_j^{(i)}, \Sigma_j, \mathcal{P}_j$)
6:         $SU\ j$ sends tuple $(\varrho_j, \Sigma_j, \mathcal{P}_j, z_j^{(i)})$ to $SU_L^{(i)}$
7:     $SU_L^{(i)}$ submits $\{(\varrho_j, \Sigma_j, \mathcal{P}_j, z_j^{(i)})\}_{j \in \mathcal{C}^{(i)}}$ to $\mathcal{BC}^{(i)}$
8:     $SU_L^{(i)}$ submits $y^{(i)}$ to $\mathcal{BC}$
```

To handle system-wise access revocations, *TrustSAS* requires that each SU's Cluster User Public Key is associated with its EPID signature over some statement that is known by all cluster members. To achieve this, each $SU\ j$ signs its Cluster User Public Key $z_j^{(i)}$ itself, which is known to all SUs in the cluster, using EPID.SIGN with its EPID Membership Private Key, sk_j (Alg 2, step 4). This serves to create a cryptographic binding between SU's EPID signature and its Cluster User Public Key. This binding will then have to be submitted as a transaction to be included in $\mathcal{BC}^{(i)}$. This is done by making SU sign the binding from the previous step using TBLS.SIGNSHAREGEN with its Cluster User Secret Key, $x_j^{(i)}$ (Alg 2, step 5). Then each SU will send the signatures, obtained in steps 4 and 5 of Alg 2, to the leader $SU_L^{(i)}$, which will collect all these signatures and include them in $\mathcal{BC}^{(i)}$. Later, when an $SU\ j$ is detected to be malicious, the leader will add SU's Cluster User Public Key $z_j^{(i)}$ along with its EPID signature to the revocation list $\mathcal{L}$.

3) Peering with $\boldsymbol{DBs}$ Phase (Alg. 1, steps 17-30): Each cluster leader will anonymously authenticate with DBs using EPID. Once a leader is authenticated by the DBs, these DBs join the established p2p network.

During this phase, a global blockchain $\mathcal{BC}$ is also created to keep track of the key system-wise events. Only DBs and cluster leaders can participate in the validation and addition of blocks to $\mathcal{BC}$. To submit a cluster-related block for inclusion in $\mathcal{BC}$, the leaders will need to have a key that identifies them and their clusters but also could be used to verify the correctness of the submitted block. This is exactly why each leader is required to submit its Cluster Public Key, $y^{(i)}$, to $\mathcal{BC}$ to be shared with DBs and other leaders. On top of that, the leader will also share a (t_i, n_i)-TBLS signature of $y^{(i)}$ to show that the Cluster Public Key was actually generated in collaboration with members of the cluster using TBLS.DKG. The validators will validate the TBLS signature through a round of BFT consensus by verifying the signature against $y^{(i)}$.

In *TrustSAS*, an operational cluster is required to transmit a beacon for a certain duration, every $\mathcal{T}_\beta$ period, so that the cluster could be discovered by nearby joining SUs, as in [20]. $\mathcal{T}_\beta$ is a system design parameter that could be adjusted based on system dynamics and on how frequent SUs join the system. A leader $SU_L^{(i)}$ needs to request this beacon from one of the DBs and can acquire it only if it successfully proves its legitimacy to DB through EPID as depicted in steps 24-28 of Alg.1. This is achieved by creating an EPID signature of a challenge message m that DB has created for this purpose. If the EPID signature is successfully verified, DB issues a beacon to $SU_L^{(i)}$ and submits the beacon to $\mathcal{BC}$ so that it is accessible by all *TrustSAS* entities. $SU_L^{(i)}$ picks some representatives from $\mathcal{C}^{(i)}$ to transmit the beacon every $\mathcal{T}_\beta$, for a specific duration over a system control channel that is known a priori and is assumed to be reserved for this purpose.

Note that SUs in $\mathcal{C}^{(i)}$ only need to have a light copy of $\mathcal{BC}$ containing the latest state of the system including the current number of clusters and their corresponding beacons. Note also that a secure session is maintained between DBs and the leader of $\mathcal{C}^{(i)}$ as long as EPID revocation list is not updated. This is to avoid running the EPID verification protocol for every block or transaction submitted by $SU_L^{(i)}$.

B. Joining TrustSAS

As depicted in Alg. 3, when an SU desires to join *TrustSAS*, it needs to tune to the control channel and scans it to detect any beacons transmitted by any nearby cluster. Failure to detect any beacons means that either no cluster is nearby or all nearby clusters are not accepting new SUs. In either case, SU will start a new cluster and will request a beacon from one of the DBs and will itself start accepting new members, as described in Alg. 1.

When the new SU detects a beacon, it invokes the TWOWAYEPID procedure with the cluster leader to ensure that the SU is legitimate and can be allowed to join the cluster, and that the leader is also in a good standing. If the two-way verification is successful, the new SU is admitted to the cluster and will immediately request $\mathcal{BC}^{(i)}$ from the cluster leader and peer with the SUs in the cluster. Newly admitted SUs will have to wait until the next $\mathcal{T}_{epoch}$ period to be able to participate in the cluster and enjoy spectrum resources.

Note that the admission of a new SU to a cluster is also subject to interference constraints. Members of the cluster must ensure that the entry of this new SU does not lead to an aggregate interference that is harmful to higher tier users or to other SUs in the cluster to satisfy coexistence. This could be resolved by adjusting grants and transmission parameters of the other SUs in the cluster, or simply denying the entry of a new SU to the cluster in the extreme case. These scenarios could be enforced by the cluster leader and agreed upon through consensus among members of the cluster.

Clusters will also need to perform rekeying operation when new SUs are added to their clusters, and this takes place at the end of each $\mathcal{T}_{epoch}$ period, where again $\mathcal{T}_{epoch}$ is a system design parameter that could be adjusted. Clusters could also

Algorithm 3 Join $\mathcal{C}^{(i)}$

```
1: $SU$ scans control channel for beacons in $\mathcal{B}$
2: if a beacon $\beta^{(i)}$ of $\mathcal{C}^{(i)}$ is found then
3:     $SU$ requests to join $\mathcal{C}^{(i)}$
4:     $v \leftarrow$TwoWayEPID$(SU, SU_L^{(i)})$
5:     if $v == True$ then
6:         $SU$ is added to $\mathcal{C}^{(i)}$
7:         $SU$ peers with $SU$s in $\mathcal{C}^{(i)}$ and downloads $\mathcal{BC}^{(i)}$
8:         $SUs \in \mathcal{C}^{(i)}$ run Rekeying$(\mathcal{C}^{(i)})$ in next $\mathcal{T}_{epoch}$
9: else
10:     $SU$ forms new $\mathcal{C}^{(i)}$ and becomes a leader $SU_L^{(i)}$
11:     $SU_L^{(i)}$ requests $\beta^{(i)}$ as in Steps 24-30 of Alg. 1
```

choose to perform rekeying when malicious and/or faulty SU s are detected. The rekeying steps are shown in Alg. 2.

C. Querying for Spectrum Availability

We now focus on describing the different steps required to privately query DBs for spectrum availability in a specific cluster. These steps are also summarized in Alg. 4.

Algorithm 4 Private Spectrum Query

```
1: $SU_L^{(i)}$ expresses interest to query $DB$s
2: $DB$s send an EPID challenge $m$ to $SU_L^{(i)}$
3: $SU_L^{(i)}$: EPID.Sign$(sk_L, \mathcal{K}_{pk}, m, \mathcal{L})$
4: $SU_L^{(i)}$ requests other $\tau - 1$ $SU$s to EPID.Sign $m$
5: $SU_L^{(i)}$ sends $\tau$ EPID signatures of $m$ to $DB$s
6: $DB$s verify the $\tau$ signatures with EPID.Verify()
7: if any signature is not valid then
8:     $DB$ adds $SU_L^{(i)}$ to $\mathcal{L}$; break
9: if $SU$s $\in \mathcal{C}^{(i)}$ experience timeout from $SU_L^{(i)}$ then
10:     $SU$s $\in \mathcal{C}^{(i)} \setminus \{SU_L^{(i)}\}$ elect new leader $SU_L^{(i)*}$
11:     $SUs \in \mathcal{C}^{(i)} \setminus \{SU_L^{(i)}\}$ run Rekeying()
12:     $SU_L^{(i)*}$ requests $\beta^{(i)}$ as in steps 24-30 of Alg. 1
13:     $SU_L^{(i)*}$ adds $SU_L^{(i)}$ to $\mathcal{L}$ and becomes $SU_L^{(i)}$
14:     go to Step 1
15: $SU_L^{(i)}$: $\boldsymbol{\mathcal{D}_q} \leftarrow$ BatchPIR$(DBs, \ell, t, r, s, \boldsymbol{q})$
16: $SU_L^{(i)}$ submits $\boldsymbol{\mathcal{D}_q}$ as block $\mathcal{B}_{epoch}$ to $\mathcal{BC}^{(i)}$
17: $SUs \in \mathcal{C}^{(i)}$ run BFT consensus to validate $\mathcal{B}_{epoch}$
18: $SU_L^{(i)}$ triggers smart contracts to divide resources
19: $SUs \in \mathcal{C}^{(i)}$ are assigned channels for current $\mathcal{T}_{epoch}$
```

In $TrustSAS$, the cluster leaders will be in charge of querying DBs for spectrum availability on behalf of their SU members, and a leader will query DBs only when: (i) Period allocated for using some channel(s) expires, (ii) quality of currently assigned channels degrades, or (iii) currently used channels need to be vacated (e.g., when requested by PUs).

1) Authentication and permission: In $TrustSAS$, in order for a leader to query DBs, its cluster is required to have a minimum of τ SUs, where τ is a system parameter that could be tuned depending on the desired robustness and security levels within each cluster. Therefore, before querying the DBs, a cluster leader, $SU_L^{(i)}$, needs to show that its cluster $\mathcal{C}^{(i)}$ meets this requirement by providing τ EPID signatures created by different legitimate SUs over a challenge message m that DBs created for this purpose; this is depicted in steps 2-5 of Alg. 4. Note that EPID prevents $SU_L^{(i)}$ from forging these τ signatures without being detected. Also, $TrustSAS$ will not require these τ EPID signatures later unless a change in the membership of $\mathcal{C}^{(i)}$ takes place. If this verification is successful, then $SU_L^{(i)}$ proceeds with querying DBs for available channels. Otherwise, DBs will label $SU_L^{(i)}$ as malicious and add it to the revocation list, $\mathcal{L}$. To ensure robustness against a leader's failures, a timeout period could be considered beyond which if the SU members do not receive spectrum availability information from their leader, the leader would be labeled as malicious and added to the revocation list, $\mathcal{L}$, and a new leader will be elected. The Rekeying procedure is then run among the cluster members, and the new leader will request a new beacon for the cluster as in steps 24-30 of Alg.1.

2) Spectrum querying: To enable private querying of DBs, $TrustSAS$ adopts multi-server private information retrieval (PIR) protocol [21], termed BatchPIR, which leverages the multiple DBs, inherently available by SAS design, to enable the cluster leaders to efficiently retrieve data records from DBs while preventing DBs from learning anything about the records being retrieved. It guarantees information-theoretic privacy, i.e. privacy against computationally unbounded servers, to cluster leaders as long as the spectrum database content, $\boldsymbol{\mathcal{D}}$, is replicated among $\ell \geq 2$ non-colluding DBs [12]. The main idea consists of decomposing each leader's query into several sub-queries each processed by a different DB to prevent leaking any information about the queried record. BatchPIR also supports batching of the queries, i.e. retrieving multiple blocks simultaneously, which is a desirable feature for $TrustSAS$. It takes as input the list of DBs, the maximum allowed number of colluding servers, the dimensions of $\boldsymbol{\mathcal{D}}$, and the indices of records of interest. For this, we assume that leaders can learn the index of the records of interest through an inverted index mechanism agreed upon with DBs.

A cluster leader, $SU_L^{(i)}$, collects queries from the SUs in its cluster $\mathcal{C}^{(i)}$, batches them together, and invokes BatchPIR with its peered DBs. $SU_L^{(i)}$ then submits the query response, $\boldsymbol{\mathcal{D}_q}$, as a block $\mathcal{B}_{epoch}$ for inclusion in $\mathcal{BC}^{(i)}$. SUs in $\mathcal{C}^{(i)}$ run BFT consensus to validate this $\mathcal{B}_{epoch}$ by simply verifying the digitally signed database records against the public key of DBs. This is to prevent the leader from maliciously sharing altered availability information.

Each record in DBs contains a smart contract that defines its usage rules. Once $\mathcal{B}_{epoch}$ is validated by SUs and added to $\mathcal{BC}^{(i)}$, the scripts of the included smart contracts will reside in $\mathcal{BC}^{(i)}$. $SU_L^{(i)}$ will issue a transaction to trigger the execution of these smart contracts, which will take as input the list of SUs in the cluster, their cell indices, and the spectrum availability information. All this information is already stored in $\mathcal{BC}^{(i)}$ and is accessible by all SUs in $\mathcal{C}^{(i)}$. Once triggered, these smart contracts run independently and automatically in a prescribed and deterministic fashion on every SU's copy of

$\mathcal{BC}^{(i)}$, in accordance with the data that was enclosed in the triggering transaction. The execution of these smart contracts will result in the automatic assignment of spectrum resources in a way that follows $TrustSAS$'s guidelines while ensuring coexistence between SUs. This assignment will be valid for the duration of the $\mathcal{T}_{epoch}$ period.

D. Notifying about Spectrum Usage

Algorithm 5 Spectrum Usage Notification

```
1: $SU_L^{(i)}$ constructs block $\mathcal{B}_i$ with usage information
2: $SU_L^{(i)}$ sends $\mathcal{B}_i$ to $SU$s in $\mathcal{C}^{(i)}$ for validation and signing
3: $(\mathcal{B}_i, \sigma_{\mathcal{B}_i}) \leftarrow$TBLS.SignReconstruct$(\mathcal{H}_i, L_1, \cdots, L_n)$
4: $SU_L^{(i)}$ submits $(\mathcal{B}_i, \sigma_{\mathcal{B}_i})$ to $\mathcal{BC}$
5: $\mathcal{V}$:$val \leftarrow$TBLS.GroupSignVerif$(\mathcal{B}_i, \sigma_{\mathcal{B}_i}, y^{(i)})$ w/ BFT
6: if $val == True$ then
7:     $\mathcal{B}_i$ is added to $\mathcal{BC}$
8:     $DB$s update their records
9: else
10:    $DB$s flag $SU_L^{(i)}$ as malicious
11:    $SU_L^{(i)}$ is added to revocation list $\mathcal{L}$ in $\mathcal{BC}$
12:    $DB$s remove $\beta^{(i)}$ from list of valid beacons on $\mathcal{BC}$
```

Once spectrum resources are allocated among SUs, the leader $SU_L^{(i)}$ shares with the DBs the allocation information, including the channels to be used by the members of $\mathcal{C}^{(i)}$, the locations where these channels will be used, and aggregated transmit power over those chosen channels. The leader can also collect the received signal strengths in the used and adjacent frequencies, the received packet error rates, and other standard interference metrics for all SUs in the cluster. The leader will propose a block $\mathcal{B}_i$ containing this information to the members of the cluster for validation. They will verify the correctness of this information and sign the block using TBLS. If the validators successfully verify that $\mathcal{B}_i$ was agreed upon and signed by members of $\mathcal{C}^{(i)}$ via BFT consensus combined with TBLS, then $\mathcal{B}_i$ is added to $\mathcal{BC}$ and DBs will include this information in their records. Otherwise, $SU_L^{(i)}$ will be flagged as malicious and its EPID signature of $y^{(i)}$ will be added to $\mathcal{L}$. These steps are summarized in Alg. 5.

IV. Security Analysis

1) Threat Model: *TrustSAS* assumes that DBs are *honest-but-curious*, in that they act "honestly" and follow the protocol in terms of handling queries and sharing spectrum availability information, but they are also "curious" about SUs' information and try to infer it from the messages they receive from SUs. *TrustSAS* also assumes that these DBs do not collude with each other, nor with the SUs. We refer to a SU that faithfully follows the protocol as *honest*; otherwise, it is referred to as *Byzantine*. *TrustSAS* assumes that these Byzantine SUs do not collude with DBs, and for each cluster $\mathcal{C}^{(i)}$, at least t_i of the n_i SUs participate in the signature, and no more than $f_i = (n_i - t_i)$ SUs are Byzantine. These t_i SUs serve as witnesses for the cluster to make sure that the leader does not communicate compromised information.

2) Security Objectives: Given the above threat model, $TrustSAS$ aims to achieve the following security objectives:

- *Private Spectrum Availability Querying:* SUs can query DBs *privately*, without revealing their operational information.
- *Private Spectrum Usage Notification:* SUs can notify DBs about their channel usage and transmission parameters *privately*, without revealing their operational information.
- *Robustness to Failures:* All security guarantees are maintained, even when a system entity fails or is compromised.
- *Immutable Public Log for Auditability:* A globally consistent, tamper-resistant log is maintained, where each system event, once produced and logged, cannot be altered or deleted.
- *Anonymity and Membership Verifiability:* SUs' authenticity can be verified before the SUs are granted system access, and SUs cannot be identified at any stage of protocol execution.
- *Location Privacy Protection of SUs:* SUs' physical location information is kept private at all times from all DBs.

3) Security Results: All proofs of the security results stated in this section are omitted here due to space limitation, and can be provided if and when requested.

Corollary 1. *$TrustSAS$ achieves unforgeability and robustness of* TBLS *signatures against an adversary that can corrupt any $f_i < n^{(i)}/2$ SUs within a cluster $\mathcal{C}^{(i)}$ as long as the Gap-Diffie-Hellman problem is intractable.*

Corollary 2. *TrustSAS ensures consistency and resistance to fork attacks for a permissioned blockchain $\mathcal{BC}^{(i)}$ running BFT consensus in every $\mathcal{C}^{(i)}$ if $t_i \geq 2f_i + 1$, where t_i is the number of signature shares required to construct a group signature for $\mathcal{C}^{(i)}$, and f_i satisfies $n_i \geq 3f_i + 1$ for BFT mechanisms [22].*

Corollary 3. *TrustSAS guarantees unforgeability and robustness of* TBLS *signatures while ensuring consistency and resistance to fork attacks for $\mathcal{BC}^{(i)}$ of $\mathcal{C}^{(i)}$ against an adversary that can corrupt any $f_i < n^{(i)}/3$.*

Corollary 4. *TrustSAS guarantees SUs with information-theoretic, private spectrum availability querying from DBs.*

Corollary 5. *$TrustSAS$ guarantees anonymous membership verification through* EPID *as long as the Decisional Diffie-Hellman and the strong RSA assumptions hold and the underlying primitives they use are secure.*

Corollary 6. *$TrustSAS$ is robust against Byzantine failures of both DBs and SUs alike.*

Corollary 7. *$TrustSAS$ guarantees location privacy information protection to all SUs.*

V. Performance Evaluation

We assess the effectiveness of $TrustSAS$ by evaluating the performance of its building blocks and algorithms. These evaluations are performed both analytically and empirically via either implementations or benchmarking of the underlying math and crypto operations using MIRACL library [23]. Experiments are carried out on a testbed that we built on Geni platform [24] using percy++ library [25]. The testbed consists of 7 VMs deployed on different Geni sites, each playing the role of a DB, and a Lenovo Yoga 3 Pro laptop with 8 GB RAM running Ubuntu 16.10 with an Intel Core m Processor 5Y70 CPU 1.10 GHz to play the role of a cluster leader.

*1) Distributed Key Generation (*DKG*):* Running DKG requires performing a number of elliptic curve point multiplications that is proportional to the number of SUs within the cluster. Using the benchmarking results that we derived with the MIRACL library [23], we provide in Table I an estimate of the average processing time experienced by each SU when running DKG. In terms of communication overhead, DKG requires 2 rounds of broadcasts, yielding $\mathcal{O}(n_i)$ messages per SU, or $\mathcal{O}(n_i^2)$ messages per cluster $\mathcal{C}^{(i)}$, when assuming no faulty SUs. Despite its relatively high cost, DKG presents no bottleneck to the system, as it is only executed at initialization or when group membership changes occur.

TABLE I
TBLS OVERHEAD WITHIN CLUSTER $\mathcal{C}^{(i)}$

Operation	Analytic Cost	Empirical Cost
DKG Computation	$\mathcal{O}(n_i) \cdot PM$	1.05 s
DKG Communication	$\mathcal{O}(n_i)$ messages	$\propto$ 1000 messages
SIGNSHAREGEN	1 $Hash$ + 1 $Expp$	0.63 ms
SIGNSHAREVERIF	$2t_i \cdot TPO$	2.3 ms
Signature size	64 bytes	64 bytes
Private key size	32 bytes	32 bytes
SIGNRECONSTRUCT	$t_i \cdot (Mulpp + Expp)$	461 ms
GROUPSIGNVERIF	$2 \cdot TPO$	2.3 ms

Variables: PM: cost of an elliptic curve point multiplication. $n_i = 1000$ SUs, $t_i = 1000$ SUs. TPO is the cost of one tate pairing. $Expp$ and $Mulpp$ are the cost of a modular exponentiation and multiplication, respectively, over modulus p.

*2) Threshold Signature (*TBLS*):* Table I provides the analytical and empirical cost of the different TBLS operations executed by SUs in $\mathcal{C}^{(i)}$. SUs repeatedly sign the consensus statement at each BFT round within the cluster. From an SU's perspective, this is relatively fast, as it involves signing a single message whose cost is dominated by a modular exponentiation operation, as shown in Table I. The leader, $SU_L^{(i)}$, will, however, incur most of the overhead, as it needs to verify all the signature shares coming from the t_i signing SUs of $\mathcal{C}^{(i)}$, before multiplying them to construct $\mathcal{C}^{(i)}$'s signature. These are the most expensive operations involved in TBLS as they require a number of modular multiplications and exponentiations that is linear in t_i as illustrated in Table I. To estimate the running time of TBLS's different operations, we use dfinity's implementation of TBLS [26].

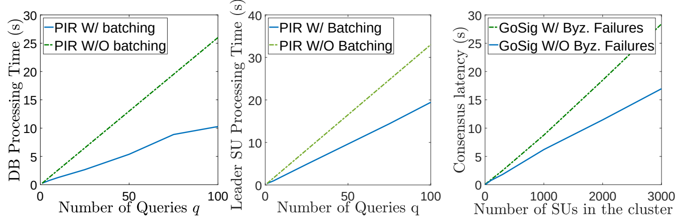
*3) Enhanced Privacy ID (*EPID*):* We evaluate EPID.SIGN and EPID.VERIFY analytically and empirically (using Intel's publicly available SDK [27]) as depicted in Table II. EPID.SIGN and EPID.VERIFY both require a number of modular exponentiations that is linear in the size of the revocations sublists; these revocation sublists are defined in [14].

Even though these delays seem relatively high, they are still reasonable, especially that these membership proof operations are independent, unfrequent, and do not occur simultaneously, once the system setup completes. Note that this proof has a linear cost in the size of the revocation list and could

TABLE II
EPID COMPLEXITY

Operation	Analytical Cost	Empirical Cost
EPID.SIGN	$(6\delta_2 + 2\delta_3 + c) \cdot Expp$	135 ms
EPID.VERIFY	$(\delta_1 + 6\delta_2 + 2\delta_3 + c) \cdot Expp$	120 ms
Signature size	257 bytes	257 bytes
Private key size	129 bytes	129 bytes

Variables: $\delta_i = |\mathcal{L}_i|$ for the revocation sublists [14] $\mathcal{L}_1$ (private key-based list), $\mathcal{L}_2$ (signature-based list), $\mathcal{L}_3$ (issuer-based list) with $\mathcal{L}_1 \cup \mathcal{L}_2 \cup \mathcal{L}_3 = \mathcal{L}$, and c is a constant. Cryptographic parameters correspond to 128-bit security level as in [28].



(a) DB PIR delay. (b) SU PIR delay. (c) GoSig BFT delay.

Fig. 2. Overhead of PIR and GoSig

become quite expensive for both signers and verifiers if such a list becomes large. One possible way to maintain a good performance of $TrustSAS$ is to impose a threshold on the list size. In this case, when the list size exceeds the threshold, FCC can create a new group and perform a rekeying operation, with each SU needing to prove to FCC that it is a legitimate member and that its membership was not revoked. This would be more efficient than carrying a large revocation list indefinitely and run expensive zero-knowledge proof operations on it. The old list will still be accessible for auditing purposes as it would have been stored already in $\mathcal{BC}$.

4) Private Information Retrieval (PIR): We use our Geni testbed to evaluate $TrustSAS$'s multi-server PIR, BatchPIR. As the obtained results in Figs. 2a and 2b show, the support of query batching by BatchPIR, which allows multiple blocks to be retrieved simultaneously, reduces the overhead at both DBs' and cluster leaders' sides. We summarize the obtained results and the analytic estimation of the overhead in Table III.

TABLE III
MULTI-SERVER PIR OVERHEAD

Operation	Analytical Cost	Empirical Cost
Leader SU's query	$q \cdot \mathcal{O}(\ell^2 r) \cdot add_{\mathbb{F}}$	4.86 s
DB processing	$q^{0.8} \cdot (\frac{8}{3} add_{\mathbb{F}} + mul_{\mathbb{F}}) \cdot rs$	2.66 s
Communication	$q \cdot (r + s)$	25 MB

Variables: $\ell = 7$: number of DBs, $q = 25$: number of batched PIR queries. DB size is $\eta = 560$ MB, s: number of field $\mathbb{F}$ elements per row, $add_{\mathbb{F}}$ and $mul_{\mathbb{F}}$ denote the cost of an $\mathbb{F}$ addition and an $\mathbb{F}$ multiplication. In a field $\mathbb{F}$ of characteristic 2, additions are equivalent to XOR and multiplications are equivalent to AND.

5) BFT Consensus: Table V shows that the communication overhead of BFT expressed in terms of number of messages sent every consensus round is quasi-linear in the size of the cluster, n_i, which translates into a total communication overhead of $\mathcal{O}(n_i^2 \log n_i)$. In this experiment, we also set the throughput between the nodes to 10 $Mbps$ and the propagation delay among SUs to 20 ms and simulate the protocol to estimate the time it takes to reach a consensus over a block.

TABLE IV
END-TO-END DELAY OF *TrustSAS* ALGORITHMS

Algorithm	Major Operations	Total Cost
Alg. 2 Rekeying within $\mathcal{C}^{(i)}$	DKG + TBLS.SIGNSHAREGEN + EPID.SIGN + BFT(n_i)	$77.47\ s$
Alg. 3: Join $\mathcal{C}^{(i)}$	TWOWAYEPID + REKEYING	$78.12\ s$
Alg. 4: Private Spec. Query	EPID.SIGN + τ EPID.VERIFY + BATCHPIR + BFT(n_i)	$13.15\ s$
Alg. 5: Spec. Usage Notifica.	t(TBLS.SIGNSHAREGEN+TBLS.SIGNSHAREVERIF)+TBLS.SIGNRECONSTRUCT+BFT(ℓ+n_c)	$1.85\ s$

Parameters: $n_i = 1000$, $t = n_i/2$, $n_c = 50$ $\ell = 7$, $\tau = 10$, bandwidth = $10Mbps$, $\eta = 560MB$, $r = 10^6$. BFT(x): one round of BFT among x parties.

TABLE V
BFT COMPLEXITY

Operation	Analytical Cost	Empirical Cost
Communication per user	$\mathcal{O}(n_i \log n_i)$	$\propto$ 3000 messages
Consensus w/o failures	$\mathcal{O}(n_i^2 \log n_i)$	$4.3\ s$
Consensus w/ failures	$\mathcal{O}(n_i^2 \log n_i)$	$6.32\ s$

Parameters: $n_i = 1000$, bandwidth = $10Mbps$, 1 signature verification per SU.

Our results, depicted in Fig. 2c, show that even for a cluster of size as large as 1000 SUs, a consensus is reachable in less than 7 s even if up to $1/3$ of the SUs are Byzantine. The overhead of BFT depends heavily on the number of participants and the number of signature verifications required by each participant. Therefore, BFT will have a different cost for each of $TrustSAS$'s algorithms. For instance in REKEYING, BFT will take as long as $76\,s$ since each SU will need to verify the signatures of all other SUs in $\mathcal{C}^{(i)}$ included in the block submitted by the leader at step 7 of Alg. 2.

6) End-to-end Delay: We provide in Table IV the end-to-end delays caused by $TrustSAS$'s different algorithms, ignoring Byzantine faultiness for simplicity. Observe that the REKEYING has the highest cost, which is invoked mainly when a membership change occurs. One way to address this is by setting REKEYING frequency small, and have joining SUs wait a little longer before they join the system. Another way to further reduce the cost in most of these algorithms is by using different quorums of users every BFT round. This will reduce the overhead but will also impact the security guarantees and robustness against failures. Despite the relatively high cost of these algorithms, note that these operations are expected to be invoked only every few hours, as it is the case for TVWS, which requires SUs to query DBs every 24 hours.

VI. CONCLUSION

We propose $TrustSAS$, a trustworthy framework for SAS that preserves SUs' operational privacy while adhering to regulatory requirements mandated by FCC in the 3.5 GHz CBRS band. $TrustSAS$ achieves this by synergizing state-of-the-art cryptographic mechanisms with the blockchain technology. We show the privacy benefits of $TrustSAS$ through security analysis, simulation and experimentation.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under NSF awards CNS-1162296 and CNS-1652389.

REFERENCES

[1] FCC, "Report and order and second further notice of proposed rulemaking, number 15-47, gn docket no. 12-354. FCC," April 2015.
[2] ——, "Order on reconsideration and second report and order, number 16-55, gn docket no. 12-354. FCC," May 2016.
[3] Y. Ye, D. Wu, Z. Shu, and Y. Qian, "Overview of lte spectrum sharing technologies," *IEEE Access*, vol. 4, pp. 8105–8115, 2016.
[4] V. Chen, S. Das, L. Zhu, J. Malyar, and P. McCann, "Protocol to access white-space (paws) databases," Tech. Rep., 2015.
[5] M. A. Clark and K. Psounis, "Trading utility for privacy in shared spectrum access systems," *IEEE/ACM Transactions on Networking (TON)*, vol. 26, no. 1, pp. 259–273, 2018.
[6] P. Marshall, *Three-tier Shared Spectrum, Shared Infrastructure, and a Path to 5G*. Cambridge University Press, 2017.
[7] W. I. Forum, "Cbrs communications security technical specification, winnf-15-s-0065," April 2017.
[8] ——, "Cbrs threat model technical report, winnf-15-p-0089," May 2016.
[9] M. Grissa, B. Hamdaoui, and A. A. Yavuza, "Location privacy in cognitive radio networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1726–1760, 2017.
[10] B. Khalfi, B. Hamdaoui, and M. Guizani, "Airmap: Scalable spectrum occupancy recovery using local low-rank matrix approximation," in *Global Communications Conference (GLOBECOM), 2018 IEEE.*
[11] M. Grissa, B. Hamdaoui, and A. A. Yavuz, "Unleashing the power of multi-server pir for enabling private access to spectrum databases," *IEEE Communications Magazine*, vol. 56, pp. 171–177, December 2018.
[12] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 1995, pp. 41–50.
[13] M. Vukolić, "Rethinking permissioned blockchains," in *Proceedings of ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, 2017.
[14] E. Brickell and J. Li, "Enhanced privacy id: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 3, pp. 345–360, 2012.
[15] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Annual International Cryptology Conference*. Springer, 1991, pp. 433–444.
[16] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Int'l Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.
[17] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Int'l Conf. on the Theory and App. of Crypto Tech.* Springer, 1999, pp. 295–310.
[18] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[19] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in *Int'l Conf. on Financial Crypto and Data Security*. Springer, 2017.
[20] T. Chen, H. Zhang, G. M. Maggio, and I. Chlamtac, "Cogmesh: A cluster-based cognitive radio network," in *2007 2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks.*
[21] W. Lueks and I. Goldberg, "Sublinear scaling for multi-client private information retrieval," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 168–186.
[22] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.
[23] M. Integer and C. Rational Arithmetic, "C++ library (miracl)," https://github.com/miracl/MIRACL, 2013, accessed: 2018-06-02.
[24] "Global environment for network innovations," https://www.geni.net/.
[25] "Percy++ library," http://percy.sourceforge.net, accessed: 2018-06-14.
[26] "Threshold bls dfinity implementation," https://github.com/dfinity/random-beacon, accessed: 2018-06-02.
[27] "The intel(r) enhanced privacy id software development kit," https://github.com/Intel-EPID-SDK, accessed: 2018-06-02.
[28] https://www.keylength.com/, accessed: 2018-06-02.