IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 30, 2021, accepted February 17, 2021, date of publication February 24, 2021, date of current version March 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3061706

Anonymous Dynamic Spectrum Access and Sharing Mechanisms for the CBRS Band

MOHAMED GRISSA[1], (Student Member, IEEE), ATTILA ALTAY YAVUZ[2], (Member, IEEE), BECHIR HAMDAOUI[1], (Senior Member, IEEE), AND CHITTIBABU TIRUPATHI[1]

[1]Electrical Engineering and Computer Science (EECS) Department, Oregon State University, Corvallis, OR 97331, USA
[2]Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA

Corresponding author: Attila Altay Yavuz (attilaayavuz@usf.edu)

This work was supported in part by the US National Science Foundation through NSF awards under Grant CNS-1162296 and Grant CNS-1917627.

ABSTRACT The Federal Communications Commission (FCC) has released the 3.5 GHz (3550-3700 MHz) band, termed Citizens Broadband Radio Service (CBRS), for shared broadband use between incumbent federal and secondary users through dynamic and opportunistic spectrum access. FCC requires that this band be operated and managed through the use of spectrum access systems (*SAS*s), which are to be deployed specifically for this purpose. The challenge is that *SAS* requires that secondary users provide some of their private operational data, such as their physical location, identity and spectrum usage, in order for them to acquire spectrum availability information. In this paper, we propose a privacy-preserving *SAS* framework, *TrustSAS*, that synergizes state-of-the-art cryptographic mechanisms with blockchain technology to enable anonymous access to *SAS* by protecting users' privacy while still complying with FCC's regulatory design requirements and rules. We evaluate the performance of *TrustSAS* through theoretic analysis, computer simulation and testbed experimentation, and show that it can offer high security guarantees, making it suitable for *SAS* environments without needing to compromise private information of its secondary users.

INDEX TERMS Blockchain, Citizens Broadband Radio Service, operational privacy, spectrum access system, spectrum databases.

I. INTRODUCTION

The Federal Communications Commission (FCC) continues its effort to promote dynamic and opportunistic access to spectrum resources, and has recently promulgated, in its Report and Order [1], the creation of the Citizens Broadband Radio Service (CBRS) in the 3.5 GHz band (3550 - 3700 MHz). This opens up previously protected spectrum used by the US Navy and other Department of Defense (DoD) members to enable spectrum sharing between government incumbents and commercial systems. In its CBRS report [1], [2], FCC prescribes the use of a centralized spectrum access system (*SAS*) to enable and govern the sharing of the CBRS spectrum among incumbent (or primary) users and CBRS (or secondary) users. Like the case of TV white space (TVWS) access, *SAS* comprises multiple geolocation spectrum databases (*DB*s) operated by different *SAS* administrators. These *DB*s are required to communicate amongst themselves to assure consistent and accurate frequency use information across one another. Also, like in the case of TVWS access, *SU*s seeking to obtain spectrum resources need to query *SAS* using their exact location information to be able to learn about spectrum opportunities in their vicinity.

A typical *SAS* supports a three-tiered access model, with three types of users: primary users (*PU*s), priority access license (PAL) users, and general authorized access (GAA) users. *PU*s are top/first tier users with the highest priority, while new CBRS users, considered as secondary users, operate either at the second tier as priority access license (PAL) users or at the third tier as general authorized access (GAA) users [3]. PAL users are assigned through a competitive auction process and have priority over GAA users. They are, however, required to vacate the spectrum upon the return of *PU*s. GAA users, on the other hand, operate opportunistically, in that they need to query *SAS* to learn about which portions of the spectrum are vacant—not being used by higher tier (*PU* or PAL) users. Even though both PAL and GAA

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

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 *DB*s to learn spectrum availability; PAL users acquire spectrum access via bidding.

For completeness, we next list some of the key design requirements that FCC has imposed for *SAS* in the 3.5 GHz band.

A. KEY SAS REQUIREMENTS

As stipulated by FCC [1], *SAS* will have capabilities and responsibilities that exceed those of TVWS databases [6]. It is projected to be more dynamic, responsive and generally capable of supporting a diverse set of operational scenarios and heterogeneous networks [7]. While some of FCC's requirements and rules for are similar to TVWS systems, other requirements are only specific to *SAS*, which include [2]:

- **Information gathering and retention**: *SAS* administrators are required to maintain accurate data about current frequency usage at all time and in all different locations. To meet this requirement, *SU*s must notify *SAS* with their current operating parameters and the channels they intend to use upon gaining knowledge of available frequencies.
- **Coexistence**: This is to prevent interference among the three tiers of users and assure a stable spectral environment for commercial operations in the CBRS band [2], [8].
- **Auditability**: *SAS* must maintain audit logs of all operations and events taking place in the system [9], including write operations to *DB*s, users' membership changes, etc. These logs are used to verify system identities' compliance with regulatory rules and policies.

It is therefore of paramount importance to incorporate and meet these requirements when designing *SAS*. The challenge, however, is that meeting such requirements presents great privacy risks to *SU*s and, as a result, may impact the adoption of this promising technology. We next discuss such privacy risks.

B. SUs' PRIVACY ISSUES IN SAS

There is a subtle privacy concern that arises with *SAS*, which pertains merely to the fact that *SU*s are required to share sensitive operational information with *DB*s in order for them to be able to learn about spectrum opportunities in their vicinity [2]. This information, which may include *SU*s' sensitive data, such as their locations, identities, spectrum usage metadata, and transmission parameters, may be collected by an adversary or a malicious *SAS* administrators (also referred to as service provider throughout) and exploited for economic, political, or other purposes [10]. For instance, fine-grained location information can easily reveal other personal information about *SU*s including their behavior, health condition, personal habits or even beliefs, especially when combined with publicly available information [11]. Things could even get worse when users are required to reveal their true identities, as it is the case in *SAS*, which could lead to more serious privacy infringements as the gained knowledge can be linked to specific individuals. Moreover, revealing spectrum usage information can give a compromised *SAS* operators full access to the spectrum usage habits, device type, times of operation, and mobility information, just to name a few.

It will not be acceptable for most users to expose such a sensitive information, especially in the presence of malicious entities that are eager to exploit this information for malicious purposes [12]. 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 privacy-preserving mechanisms that protect *SU*s' sensitive information while at the same time abiding by FCC's rules and policies prescribed for *SAS*.

As most of these rules require *SU*s to share a great deal of sensitive information, they seem to be conflicting with *SU*s' privacy objective. 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 *SU*s so as to promote this new spectrum sharing technology. This dilemma makes the task of designing *SAS* mechanisms that provide privacy guarantees to *SU*s, while allowing them to use the system in compliance with *SAS*'s requirements and rules a very challenging one. 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.

C. LIMITATION OF THE STATE-OF-THE ART

Existing privacy-protection approaches have mostly focused on preserving the location privacy of *SU*s in TVWS database-driven spectrum sharing systems [11]–[16], which have different requirements compared to the new CBRS *SAS*. For instance, unlike *SAS*, these TVWS systems are not concerned with coexistence and interference protection amongst *SU*s, nor do they require *SU*s report their spectrum usage information to the databases upon determining which bands they will be using. Some of these approaches have relied on the concept of *k-anonymity*, which provides a simple way to hide the location of an *SU* by sending k queries that include the location of the querying *SU* and some $k-1$ randomly chosen other locations which do not necessarily belong to the same cluster or region. This kind of approaches offers weak privacy guarantees unless the value of k is very large, in which case it may pose practicality issues. Remark that *k-anonymity* approaches do not offer a provable security guarantee, which are necessary to precisely quantify and offer a high-level of security.

There have been some efforts that aim to preserve operational privacy for *SAS* in the 3.5 GHz band, but only for *PU*s, which are, in this case, military and governmental entities. For instance, Clark *et al.* [7] propose a general privacy framework to evaluate the privacy of *PU*s in *SAS* and model the *PU* privacy problem as an optimization problem that aims to select obfuscation strategies to protect privacy without severely reducing the utility of the shared spectrum. Bhattari *et al.* [18] propose an obfuscation strategy that can be implemented by the spectrum database to perturbate responses to *SU*s spectrum queries in order to maximize *PU*s location privacy while ensuring that the expected degradation in *SU*s access to the spectrum does not exceed a threshold. While the *PU* privacy issue is critical in *SAS*, it is, however, orthogonal to the work that we present in this paper. Privacy preserving techniques designed to protect the location information of *PU*s cannot be applied directly to protect *SU*s' location mainly because *PU*s and *SU*s do not share the same kind of information with *DB*s.

Despite the importance of providing privacy protection of *SU*s in *SAS*, only a handful of approaches were proposed to address this problem. *P2-SAS* [19] relies on multi-party computation via Paillier partial homomorphic encryption. This severely limits the type of functions that this scheme can compute over the encrypted operational data of both *SU*s and *PU*s to only some limited/basic operations. These operations might not capture the complex models required to calculate spectrum availability information. A direct use of generic secure multi-party computation (SMPC) techniques might also not be practical in SAS, since *PU*s are military and governmental entities that do not want to continuously engage in an interactive protocol with *SU*s and the spectrum database. Another limitation of SMPC solutions is that they are vulnerable to collusion among multiple parties.

PSEO in [5] requires *DB* to share an attenuation map of the whole covered region with all *SU*s, who should use it to calculate the interference that they may cause to a *PU*. This is done by trying different combinations of *PU* operational data at each possible location of the region covered by *SAS*, then encrypt this map with Paillier partial homomorphic encryption along with the public key of the *PU*, and finally sends them to the spectrum database. Each *PU* is also required to calculate an interference map by trying all possible combinations of a *SU* operational data at every single location of the covered region, then it encrypts the resulting map using its public key and Paillier cryptosystem, and sends the ciphertext to the spectrum database. The spectrum database will multiply the two ciphertexts together and sends the result back to the *PU*, who can decrypt it to compare the maps. Based on this comparison, *PU* will decide whether to allow the *SU* to use its resources. One can observe that this scheme requires a very large number of costly Paillier encryption over a large input size. Hence, it cannot scale for a large number of *SU*s with a large area coverage, especially with the fact that *SAS* is designed for highly dense areas [21], [22]. It also places a high trust on *SU*s by assuming that they will accurately and honestly calculate the interference. This overwhelms *SU*s with heavy computation, and reduces the role of *DB*s to just a simple gateway between *PU*s and *SU*s. Finally, PSEO requires that *SU*s communicate directly with *PU*s, which is not a realistic assumption as *PU*s in *SAS* systems are military entities that will not be willing to engage in an interactive protocol with all *SU*s that seek to use their spectrum.

It is also worth to note that, aforementioned approaches consider a single-service provider *SAS* architecture, which deviates from the real-world architecture proposed by the FCC. Moreover, various *SAS* specific requirements were not considered. Specifically, they focus on the spectrum sharing between *PU*s and *SU*s and do not consider the coexistence between *SU*s themselves, the spectrum usage notification and system auditability, all of which are requirements specifically introduced for *SAS* by the FCC.

Tackling this issue is challenging, mostly due to the conflicting interests of guaranteeing *SU*s' privacy protection while meeting *SAS*'s design requirements. Besides, relying on traditional privacy enhancing technologies to achieve these goals is not be possible. For instance, simply using digital signatures to satisfy the authentication requirement will obviously expose the user's identity which conflicts with the anonymity goal. On the other hand, ensuring anonymity may conflict with the authentication and the accountability requirements of *SAS* and may, as a result, lead to external adversaries gaining access and posing a serious risk to the whole *SAS*.

Next, we summarize our main contributions in this work.

D. CONTRIBUTIONS

In this paper, we propose *TrustSAS*, a trustworthy design framework that enables *SU*s to make use of spectrum opportunities in the CBRS band but without needing to compromise their privacy, while at the same time complying with FCC's requirements and rules as outlined in Section I-A. More specifically, this work, which is, to the best of our knowledge, the first to achieve these seemingly conflicting goals, makes the following contributions:

1) TRUSTWORTHY *SAS* FRAMEWORK

We observe that most of *SAS*'s design requirements conflict with the privacy of *SU*s. This led us to design *TrustSAS* which synergizes multiple cryptographic building blocks in an innovative way and leverages the unique properties of the blockchain technology to preserve *SU*s' operational privacy while meeting these requirements.

2) CLUSTERED *SAS* ARCHITECTURE

We propose to rely on a clustered system architecture that helps *TrustSAS* enjoy a reasonable overhead while offering high privacy and security guarantees for entities in *SAS*.

3) DETAILED PERFORMANCE ANALYSIS

We provide a thorough performance analysis that combines theoretical cost estimation, simulation and experimentation using implementations of the cryptographic building blocks to measure the overhead introduced by the different *SAS* components.

Improvements Over the Infocom'19 Conference Version [23] This article is the extended version of [23], which includes the following substantial improvements: (i) In this submission, we address the case where malicious secondary users may misbehave and operate on channels that are different from the ones that were assigned to them. We rely on spectrum permits that are embedded to the physical-layer signals to detect such behavior. (ii) In this submission, we provide a more detailed security analysis of the proposed framework and its different algorithms by including proofs to the previously obtained security results. (iii) In this submission, we provide a more detailed performance analysis that assesses the performance of the different algorithms and building blocks of the proposed framework. We also evaluate the impact that this framework might have on the system level. This is achieved via simulation by modelling users' join events, activities and placement within the clusters using random processes.

The remainder of this paper is organized as follows: Section II presents a high-level overview of the proposed framework. Section III provides an overview of the main building blocks used in our framework. Section IV describes the proposed framework, *TrustSAS*. The security analysis and performance evaluation of *TrustSAS* are provided in Sections V and VI, and the paper is concluded in Section VII.

II. SYSTEM AND FRAMEWORK OVERVIEW

In this section, we present the system architecture and provide a high-level overview of *TrustSAS*, the proposed design framework that preserves *SU*s' operational privacy while meeting FCC's design requirements and rules for *SAS*. Detailed description of *TrustSAS* will be provided in later sections.

A. ARCHITECTURAL COMPONENTS

As illustrated in Fig. 1, *TrustSAS* comprises three main architectural entities: FCC, multiple *DB*s, and multiple *SU*s. 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 *SU*s, and granting them permissions to join *TrustSAS*. *TrustSAS* leverages and relies on the existence of multiple *DB*s for spectrum access, each typically run by a different service provider. These *DB*s are assumed to be synchronized and to be sharing the same content, as mandated by FCC. *TrustSAS* also comprises multiple *SU*s including a set of pre-registered *SU*s, that are assumed to be deployed specifically for playing the role of anchor nodes in *TrustSAS*. These anchor *SU*s serve to establish a peer-to-peer (p2p) network for *SU*s and make it discoverable by new *SU*s. As the focus of this framework is mainly on the security and privacy mechanisms aimed at enabling anonymous access to *SAS* by protecting the privacy of *SU*s while still complying with FCC's design requirements and rules, we do not consider the impact of wireless channel reliability, and instead, assume that *SU*s have reliable wireless access to the system. This proposed *TrustSAS* is orthogonal (and hence complementary) to the new and existing approaches that can be used to mitigate the impact of the wireless channel impairments.

TrustSAS also maintains one global blockchain for the entire system, and one local blockchain for each of its clusters (see Fig. 1). Only *DB*s and cluster leaders can be validators in the global blockchain to validate and propose blocks. For the local blockchains, cluster leaders propose new blocks, and these blocks are validated by the members of the cluster. Using blockchains at the cluster as well as at the system levels enables *TrustSAS* to log and keep track of all activities and key events in a way that is available, yet immutable, to all entities in the system, thereby allowing *TrustSAS* to meet the *auditability* requirement, as we stated in Section I-A. The two levels of blockchains that we have in the system allow to separate cluster level information, which is recorded in the cluster blockchain, from system level operations and events, which are recorded in the global blockchain, for better auditability. It also helps reducing the communication and storage overheads in the system, and the amount of exposed information regarding each cluster operations and its members to other clusters and *DB*s. Cluster-level blockchains are also used as a trusted infrastructure to generate and keep track of the keys generated collaboratively by the member *SU*s which are used to sign the information that the cluster will share with the rest of the system.

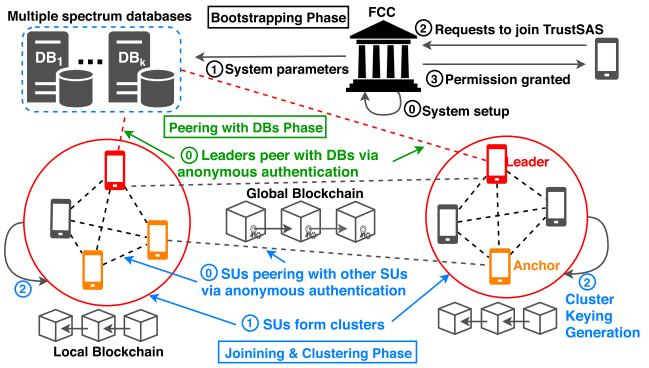


FIGURE 1. TrustSAS architecture and initial operations.

Initial system setup has three main phases (refer to Fig. 1):

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 *DB*s to define channel usage rules, such as the maximum number of *SU*s allowed to transmit simultaneously in given location over a specific channel, the maximum transmit power each *SU* is allowed to use given the current number of *SU*s present in a location, etc. Spectrum assignment within each cluster follows the constraint imposed by the smart contracts defined by the *SAS* spectrum databases specific to each channel in each location. With these smart contracts, *TrustSAS* ensures fair sharing of the spectrum resources within each cluster, and limits the interference among *SU*s. As the spectrum databases are aware of the spectrum usage information of each cluster, and they have a global view of the system-wise spectrum usage, it is up to the spectrum databases to construct the spectrum availability information in a way that also minimizes interference with *PU*s transmission and also inter-cluster interference. All of this will enable *SAS* to satisfy the *coexistence* requirement, described in Section I-A.

We further assume that each record is signed with a common private key that is shared among *DB*s so that record integrity could be verified later when fetched by *SU*s using the corresponding public key. For simplicity, we assume that channel usage is permitted over a fixed duration independently from the channel, and that *SU*s need to query *DB*s 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*' location is expressed through its cell index.

B. TrustSAS INITIAL SETUP

1) BOOTSTRAPPING PHASE

FCC needs to first create system parameters and keys, specific to *TrustSAS*, and share them with *DB*s. Also, before joining *TrustSAS*, an *SU* first needs to register and request *SAS* access privileges from FCC. Once registered, FCC provides the *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 *SU*s and the *DB*s are established over secure channels, so as to ensure that the spectrum queries are authenticated, not tampered with, and performed privately. 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.

2) JOINING AND CLUSTERING PHASE

Registered *SU*s that join *TrustSAS* will maintain communication with one another via an overlay peer-to-peer (p2p) network, and a newly joining *SU* will rely on anchor *SU*s to discover and join the p2p network. As we discuss later, *TrustSAS* relies on an anonymous digital signature technique, explained in Section III-C, to enable all these *SU*s to anonymously authenticate and verify each other's legitimacy while peering with each other. This anonymous authentication will also enable *SU*s 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 *SU*s group themselves into clusters and elect cluster leaders, with the leaders being responsible for representing their *SU*s for interacting with other system entities. Not only will this improve *TrustSAS* scalability, but also protect *SU*s' privacy, as it will limit the interaction with *DB*s to only cluster leaders.

Leader election could be achieved by making participant *SU*s generate a random number using a verifiable random function (VRF) similar to several blockchain protocols such as Algorand [24], Dfinity [25], and Ouroboros Praos [26]. If the resulting random number satisfies a certain pre-defined condition, it means that the *SU* was elected as a leader. The leader can then broadcast a block along with the associated proof generated by the VRF to the network to cryptographically prove that it was elected. This election could be triggered anytime *SU*s within the cluster, through a BFT consensus round, agree that a new leader needs to be elected. There could be several ways to incentivize *SU*s to become leaders. One possible way may be to guarantee that the leader always has access to the spectrum during resource allocation, no matter what the number of *SU*s is in the cluster, while other *SU*s will have intermittent access to the resources if these resources are overbooked.

Once clusters are established, *SU*s 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.

3) PEERING WITH *DB*s PHASE

Once clusters are formed, the leaders will anonymously authenticate with *DB*s, and upon authentication, these *DB*s will join and be part of the established p2p network. This way, *DB*s will not be involved in the initial clustering of *SU*s, and therefore they will not be able to infer the *SU*s' location information.

C. TrustSAS MAIN OPERATIONS

TrustSAS has two main operations (refer to Fig. 2):

1) QUERYING SPECTRUM AVAILABILITY INFORMATION

Each cluster leader acts on behalf of its *SU* members and privately queries *DB*s for spectrum availability information. Even though the true identities of all *SU*s, including leaders, are hidden in *TrustSAS*, this is not sufficient to preserve their operational privacy. In fact, since each record in *DB*s is associated with a unique location, *DB*s 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 private information retrieval (*PIR*) protocol [27], which is

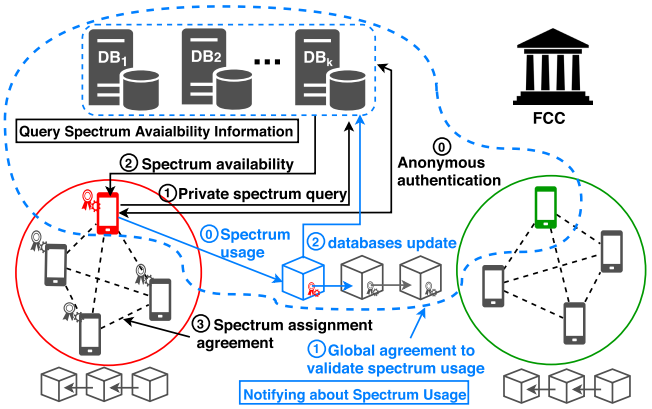


FIGURE 2. TrustSAS operations.

a mechanism that enables a user to retrieve a record from a database while preventing the database from learning any information about the identity of the record, nor that of the user; this will be explained in Section III-D. After learning the spectrum availability information, members of each cluster will distributively reach an agreement on how the spectrum resources are to be assigned among them.

2) NOTIFYING ABOUT SPECTRUM USAGE

Once a spectrum assignment agreement is reached, the cluster leader will notify the *DB*s about its cluster spectrum usage, as required by FCC. This information includes channels that are to be used, total aggregate transmission power for each channel at each cell, time of use, and other transmission parameters. It will be used by *TrustSAS* 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 IV, *TrustSAS* ensures that cluster leaders report an accurate and non-altered spectrum usage information that is easily verifiable. Other leaders and *DB*s will distributively reach an agreement about the validity of this information, which, if valid, will be included in *DB*s' records.

Please remark that the use of an overlay peer-to-peer network, the clustering of SUs into multiple clusters based on location, the use of the blockchain technology, and the key management that involves the FCC, were not part of the original design proposals for SAS systems. We have incorporated these new aspects into our design to enforce the location privacy of SUs but also abide by the requirements imposed by the FCC as we show in the remaining parts of this manuscript.

In Section III, we explain the different technologies that *TrustSAS* builds on to create a trustworthy *SAS* before we present the details of *TrustSAS* in Section IV.

III. PRELIMINARIES AND BUILDING BLOCKS

A. THRESHOLD SIGNATURES

A (t, n)-threshold signature scheme is a special type of group signatures, where n members jointly set up a group public key while each retaining an individual secret key share. At least $t+1$ out of the n members of the group are required and sufficient to create a group signature which validates against the group public key. *TrustSAS* uses the robust (t, n)-threshold BLS (TBLS) signature scheme [28], which transforms BLS [29] into a threshold signature scheme using secret sharing [30]. For completeness, we next briefly describe TBLS.

Let G be a *gap Diffie-Hellman* group of prime order p, where the *computational Diffie-Hellman* problem is hard, but the *decision Diffie-Hellman* problem is easy [29]. TBLS comprises five subroutines as illustrated in Alg. 1. The first subroutine, DKG, is the distributed key generation protocol for discrete-log based systems [31]. DKG is jointly executed by a set of n participants $\{P_1, \cdots, P_n\}$ and takes as input a global publicly available information I and outputs a group public key y. The private output of each participant P_i after the execution of DKG is x_i such that $(x_1, \cdots, x_n)$ is a (t, n)-threshold secret sharing of the private key $x = \log_g y \in \mathbb{Z}_p$. These shares are constructed using Shamir secret sharing [32] such that any subset $\mathcal{S}$ of $t+1$ participants can recover x using Lagrange interpolation $x = \sum_{i\in\mathcal{S}} L_i x_i$, where $L_i = \prod_{j\in\mathcal{S}} -z_j/(z_i - z_j)$ is the Lagrange coefficient, $z_i = g^{x_i}$ is the public key of P_i, and g is the generator of group G.

Algorithm 1 TBLS

$(y, x_1, \cdots, x_n, z_1, \cdots, z_n) \leftarrow \text{DKG}(I)$
▷ y: group public key
▷ x_i: participant P_i's private share of the group private key x
▷ z_i: public key of participant P_i, $z_i = g^{x_i}$
▷ $I \leftarrow \{g, p, H\}$: public keying information where g: a generator of G; p: order of G; $H : \{0,1\}^* \rightarrow G$ is a cryptographic hash function.

$\sigma_i \leftarrow \text{SignShareGen}(x_i, m)$
▷ σ_i: P_i's signature share; m: message to be signed

$valid \in \{True, False\} \leftarrow \text{SignShareVerif}(m, \sigma_i, z_i)$

$(m, \sigma) \leftarrow \text{SignReconstruct}(\mathcal{H}, L_1, \cdots, L_n)$
▷ $\mathcal{H}$: Set of $t+1$ honest participants.
▷ $L_1, \cdots, L_n$: Publicly known Lagrange coefficients
▷ σ: the group signature over message m

$valid \leftarrow \text{GroupSignVerif}(m, \sigma, y)$

In the SignShareGen subroutine, each P_i computes the signature share $\sigma_i = H(m)^{x_i}$ over the message $m \in \{0,1\}^*$, and broadcasts σ_i. Using SignShareVerif, any member of the group can verify that P_i is honestly calculating its share σ_i by checking that $(g, z_i, H(m), \sigma_i)$ is a *decision Diffie-Hellman* tuple, i.e. $\log_g z_i = log_{H(m)}\sigma_i$ given a message m, a signature σ_i, and a public key z_i. Any participant or set of participants can then reconstruct the group signature using SignReconstruct. For the sake of illustration, we assume this is run

by a designated participant. This participant collects $t+1$ signature shares from a set of $t+1$ participants and checks the validity of each share using SignShareVerif. If a share is not valid, a new share is requested from the corresponding participant, otherwise, it is assumed malicious. When all $t+1$ shares are valid, the complete group signature σ is recovered as $\sigma = \prod_{i\in\mathcal{H}} \sigma_i^{L_i}$, where $\mathcal{H}$ is a set of $t+1$ honest participants. Finally, the group signature is verified using GroupSignVerif against y exactly as in SignShareVerif.

Note that in TBLS, it is not necessary to reconstruct the private key during the signing process. Even after repeated signing, no one could learn any information about the private key that would enable them to create signatures without a $(t+1)$-sized group [33].

B. BLOCKCHAINS

Blockchains are a form of a shared decentralized append-only database, or ledger, maintained by a set of nodes, often mutually distrusting each other, enabling them to perform write operations to this shared log [34]. Writes to this shared log, known as transactions, are organized as *blocks* each wrapping multiple transactions into a single atomic write operation. A block typically contains a timestamp and a hash of the previous block, making each block chained to its predecessor to form a chain of blocks that are signed and secured from adversarial tampering or incidental changes.

Permissioned blockchains are a special type of blockchains that have evolved as an alternative to permissionless blockchains, used in cryptocurrencies such as Bitcoin [34], to tackle the need for running this technology among a set of known and identifiable members. They have an extra access control layer [35] and are particularly interesting in business applications where participants, not necessarily trusting one another, need a mechanism of identifying each other.

The consensus mechanism is arguably the most important component of any blockchain system since any update to the blockchain needs to be agreed upon by all participants. Otherwise, nodes will have different copies of the distributed ledger, leading to what is known as forks. Hence, the system will no longer be able to maintain a unique reliable chronology unless these forks are pruned. This becomes more challenging as nodes in a blockchain system do not trust each other. As a result, a distributed consensus mechanism that tolerates Byzantine failures is required in order to establish an agreement in the network. Distributed consensus mechanisms for permissioned blockchain systems [36] rely on purely communication-based consensus protocols, where participants usually have equal votes and go through multiple rounds of communications to reach consensus about a certain block. These protocols, known as *Byzantine fault tolerant* (BFT) protocols, require at least $3f+1$ nodes to reach consensus provided that up to f nodes are Byzantine [35].

Smart Contracts are scripts meant to perform like regular contracts and aim to automatically and securely execute obligations without reliance on a centralized enforcement authority. Each smart contract is identified by a unique address, and its code resides on the blockchain as a special transaction. As such, its code can be examined by every node in the network. Indeed, since all the interactions with a contract occur via signed messages on the blockchain, every node gets a cryptographically verifiable trace of the contract's operations [37]. They could be considered as stored procedures that get invoked and triggered whenever some conditions are met or whenever a transaction is addressed to them. Once a smart contract is triggered, it runs independently and automatically in a prescribed and deterministic fashion on every node, in accordance with the data that was enclosed in the triggering transaction. Their properties make it possible to have general purpose computations occur on the blockchain.

C. ENHANCED PRIVACY ID (EPID)

EPID [38], an extension of the *Direct Anonymous Attestation* (DAA) protocol [39], is an anonymous digital signature algorithm that enables members of a group to prove to one another their group membership legitimacy without revealing their true identities. EPID generates and uses three main keys: *(i) Membership Verification Public Key* ($\mathcal{K}_{pk}$), which is created by the *Issuer* (an entity that oversees the group), shared among all group members, and used by the *Verifier* (i.e., any group member) to anonymously verify the membership legitimacy of a user; *(ii) Membership Issuing Secret Key* ($\mathcal{K}_{sk}$), which is kept secret and used only by the *Issuer* to create a unique *membership private key* for each group member; and *(iii) Membership Private Key* (sk_U), which is created by the *Issuer* for a group member U, upon the member's request. Each member uses its membership private key to prove its membership legitimacy to the *Verifier* anonymously.

EPID also allows revocation of group members. It maintains three lists, each for supporting one type of revocation: $\mathcal{L}_{priv}$ for private key-based revocation, $\mathcal{L}_{sig}$ for signature-based revocation, and $\mathcal{L}_{iss}$ for issuer-based revocation. $\mathcal{L}_{priv}$ contains the private keys of all members whose keys have been compromised. $\mathcal{L}_{sig}$ contains the signatures of members that have been reported as misbehaving members. $\mathcal{L}_{iss}$ contains the members that have been removed by the *Issuer* (e.g., when members leave the group).

EPID typically runs four procedures, which are summarized in Algorithm 2. The first, EPID.Setup() (step 1), is run by the *Issuer* and outputs $\mathcal{K}_{pk}$ and $\mathcal{K}_{sk}$. The second, EPID.Join() (step 2), is run interactively between a new member and the *Issuer*. It results in the member obtaining a unique private key, sk_U. The third procedure, EPID.Sign() (step 3), allows a member to prove its membership anonymously to the *Verifier* by signing a challenge message m provided to it by the *Verifier*. During this procedure, the user proves to the *Verifier* that its private key and signature do not belong to the revocation lists, $\mathcal{L}_{iss}$ and $\mathcal{L}_{sig}$. The *Verifier* validates the correctness of the EPID signature of a member through the fourth procedure, EPID.Verify (step 4), using the public key,

Algorithm 2 EPID

1: $(\mathcal{K}_{pk}, \mathcal{K}_{sk}) \leftarrow \text{SETUP}(\kappa)$ ▷ κ is the security level

2: $sk_U \leftarrow \text{JOIN}(\mathcal{K}_{pk}, K_I)$ ▷ sk_U: private key of user U
▷ K_I: *Issuer* long-term public key to authenticate with users

3: $(\Sigma_U, \mathcal{P}_U) \leftarrow \text{SIGN}(sk_U, \mathcal{K}_{pk}, m, \mathcal{L}_{sig}, \mathcal{L}_{iss})$
▷ Σ_U: EPID signature of message m
▷ $\mathcal{P}_U$: Set of non-revocation proofs of user U for each element in $\mathcal{L}_{sig}$ and $\mathcal{L}_{iss}$
▷ m: The *Verifier's* message as a challenge to the user

4: $valid \in \{success, fail\} \leftarrow \text{VERIFY}(\mathcal{K}_{pk}, m, \Sigma_U, \mathcal{P}_U, \mathcal{L})$

$\mathcal{K}_{pk}$. During this process, the *Verifier* also checks that the member does not belong to any of the revocation lists.

EPID enables the *Verifier* to link any two signatures produced by the same user, allowing it to prevent any malicious *SU* from forging multiple signatures on behalf of other *SU*s. EPID relies heavily on the well-known zero-knowledge proof concept [40] to enable the signer to prove its membership without needing to share its identity or private key.

D. MULTI-SERVER PRIVATE INFORMATION RETRIEVAL (PIR)

A *PIR* protocol enables a user to retrieve a data record of its choice from a database while preventing the database from learning anything related to the item being retrieved [27]. Multi-server *PIR* is a special variant of this protocol that requires the replication of the database among $\ell \geq 2$ non-colluding servers [27], [41]. This enables it to efficiently realize information-theoretic privacy, i.e. privacy against computationally unbounded servers. The main idea consists of decomposing each user's query into several sub-queries each processed by a different server to prevent leaking any information about the item that the user is querying. Despite its efficiency and optimal privacy level, this technology has remained purely theoretical without a real-life application due to the database replication requirement. However, thanks to the inherent design of *SAS*, which contains and relies on multiple *DB*s by design, it was possible for our proposed *SAS* framework, *TrustSAS*, to benefit from such a technology.

TrustSAS is agnostic to which multi-server *PIR* protocol is used. However, for the sake of performance evaluation and security analysis, *TrustSAS* adopts the *PIR* protocol proposed by Lueks *et al.* [42]. Besides, this protocol supports batching of the queries, i.e. retrieving multiple blocks simultaneously, which is a desirable feature for our framework. It combines Goldberg's robust information-theoretic *PIR* [41] with fast matrix multiplication techniques inspired by batch codes [43].

In Goldberg's protocol the database is modeled as a matrix $\mathcal{D}$ of size $r \times s$ of elements in finite field $\mathbb{F}$. Every row in $\mathcal{D}$ corresponds to a single block in the database. To request the j^{th} block (non-privately) the user simply constructs the j^{th} standard basis vector e_j of $\mathbb{F}^r$ (which is a vector of length r with all zeros except for the j^{th} position having a value 1) then sends it to the server. The server then calculates the product $e_j \cdot \mathcal{D}$ to obtain the requested j^{th} block. To make the query private, the user creates ℓ Shamir secret shares [32] of e_j and sends one share to each of the ℓ servers which compute the product with $\mathcal{D}$ and return the result. The user then uses Lagrange interpolation (explained in Section III-A) of the results to recover the j^{th} block. Since Lagrange interpolation of the shared query vectors yields e_j and due to the linearity of matrix multiplication and Lagrange interpolation, then interpolation of the results yields the desired block. The *t-out-of-*ℓ Shamir secret sharing ensures that as long as at most t servers collude, they learn nothing about the desired block. This protocol is also robust against Byzantine servers thanks to its reliance on error-correction codes. Batch queries are supported in [42] through batch codes [43]. We provide the parameters required to run this protocol in Alg. 3, which we will use later

Algorithm 3 Private Information Retrieval

1: $\mathcal{D}_q \leftarrow \text{BATCHPIR}(DBs, \ell, t, r, s, \boldsymbol{q})$
▷ $\mathcal{D}_q$: retrieved records from $\mathcal{D}$
▷ DBs: database servers; ℓ: number of DBs
▷ t: number of servers that can collude (i.e. privacy level)
▷ r: number of rows; s: number of words per row
▷ $\boldsymbol{q}$: query vector for multiple blocks

IV. THE PROPOSED FRAMEWORK: *TrustSAS*

We now present *TrustSAS*, the proposed trustworthy *SAS* that preserves *SU*s' operational privacy but while ensuring that *SAS* entities abide by the FCC requirements. *TrustSAS* comprises several algorithms, each will be discussed separately in this Section, synergizing the different technologies that we presented in Section III in an innovative way to achieve the aforementioned goals. For convenience, we summarize the notations used in the remaining parts of this paper in Table 1. Also, for simplicity, throughout the description of *TrustSAS*, whenever some entity submits a block to a blockchain, that implicitly implies that the entity signed the block and that the block was validated via BFT consensus by the validators before it was added to the blockchain.

A. SYSTEM SETUP

The first component of *TrustSAS*, depicted in Alg. 4, consists of setting up the system parameters and the required keys at initialization, as described next.

1) BOOTSTRAPPING PHASE (Alg. 4, STEPS 2-10)

In *TrustSAS*, FCC plays the role of the EPID *Issuer*, described in Section III-C, and establishes a permissioned-group for *TrustSAS*. It generates and provides two keys. The first key, EPID Membership Verification Public Key, $\mathcal{K}_{pk}$,

TABLE 1. Notations.

κ	Security level
$\mathcal{K}_{pk}$	Membership Verification Public Key
sk_{SU}	SU's EPID membership private key
K_{FCC}	FCC long-term public key
$\mathcal{A}$	Set of anchor SUs
$\mathcal{C}^{(i)}$	i^{th} cluster in the system
$\mathcal{R}^{(i)}$	Set of representatives from $\mathcal{C}^{(i)}$ to transmit β_i
$\mathcal{BC}$	global blockchain
$\mathcal{BC}^{(i)}$	Local blockchan of cluster $\mathcal{C}^{(i)}$
$\mathcal{T}_{epoch}$	Validity period of spectrum availability and clusters' TBLS keys
$\mathcal{B}^{(i)}$	A block proposed by members of $\mathcal{C}^{(i)}$
$\mathcal{H}_i$	A set of $t_i + 1$ honest SUs within $\mathcal{C}^{(i)}$ required for signing
n_i	Number of SUs within cluster $\mathcal{C}^{(i)}$
$n_{\mathcal{C}}$	Number of clusters in the SAS
ℓ	Number of DBs in the SAS
$\mathcal{T}_{\beta}$	Beacons are transmitted periodically every $\mathcal{T}_{\beta}$
$y^{(i)}$	Cluster $\mathcal{C}^{(i)}$'s TBLS public key
$x^{(i)}$	Cluster $\mathcal{C}^{(i)}$'s TBLS secret key
$z_j^{(i)}$	SU j TBLS public key in $\mathcal{C}^{(i)}$

is shared among and used by all *TrustSAS* members/entities to anonymously authenticate and verify independent signatures among each others, using the EPID.Verify procedure. The second key is the EPID Membership Private Key, sk_{SU}, provided to the SU so it can use to prove its membership legitimacy to other *TrustSAS* entities. Each SU will have its own EPID Membership Private Key.

Initially, some SUs will be appointed to serve as anchor nodes, and will run the TwoWayEPID subroutine (Alg. 4, step 1) among themselves as a way to authenticate each other anonymously before they peer up and initiate the overlay p2p network. An SU seeking to join *TrustSAS* needs to initiate a join request to FCC via the EPID.Join procedure (Alg. 4, step 9). This step allows SU to obtain $\mathcal{K}_{pk}$ and sk_{SU}, as well as the list of anchor nodes, denoted by $\mathcal{A}$ throughout. Note that this process creates a *many-to-1* asymmetric relationship between $\mathcal{K}_{pk}$ and all sk_{SU}s.

2) PEERING AND CLUSTERING PHASE (Alg. 4, STEPS 11-16)

Once the joining SU obtains the list, $\mathcal{A}$, of anchor SUs from FCC, it uses it to discover and join the ongoing p2p network. The joining SU then needs to authenticate with its peers and verify their legitimacy via TwoWayEPID (Alg. 4, 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. We can add an extra privacy layer to protect these locations by exposing them to only SUs that are potential cluster members by adapting the private proximity testing protocol due to Narayanan *et al.* [44].

The members of each $\mathcal{C}^{(i)}$ will maintain a cluster (local) blockchain, $\mathcal{BC}^{(i)}$, to log and keep track of key events taking place in the cluster. Also, members of each $\mathcal{C}^{(i)}$ will have to run steps 2-6 of the Rekeying operation described in Alg 5, to jointly generate the keys required for performing

Algorithm 4 *SAS* Setup

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: $(\Sigma_A, \mathcal{P}_A)$ ←EPID.Sign(sk_A, $\mathcal{K}_{pk}$, m_B, $\mathcal{L}_{sig}$, $\mathcal{L}_{iss}$)
B: v_A ←EPID.Verify($\mathcal{K}_{pk}$, m_B, Σ_A, $\mathcal{P}_A$, $\mathcal{L}$)
B: $(\Sigma_B, \mathcal{P}_B)$ ←EPID.Sign(sk_B, $\mathcal{K}_{pk}$, m_A, $\mathcal{L}_{sig}$, $\mathcal{L}_{iss}$)
A: v_B ←EPID.Verify($\mathcal{K}_{pk}$, m_A, Σ_B, $\mathcal{P}_B$, $\mathcal{L}$)
return $v_A \wedge v_B$

Bootstrapping phase

FCC: $(\mathcal{K}_{pk}, \mathcal{K}_{sk})$ ← EPID.Setup(κ)
FCC shares $\mathcal{K}_{pk}$ with DBs
$(sk_{SU}, \mathcal{K}_{pk})$←EPID.Join($\mathcal{K}_{pk}$,$\mathcal{K}_{FCC}$) $\forall SU \in \mathcal{A}$
for SU $k \in \mathcal{A}$ **do**
  for SU $l \in \mathcal{A} \setminus \{k\}$ **do**
    TwoWayEPID(k, l)
All SUs $\in \mathcal{A}$ peer up with each other
Joining SU: $(sk_{SU}, \mathcal{K}_{pk})$ ← EPID.Join($\mathcal{K}_{pk}$, $\mathcal{K}_{FCC}$)
FCC shares $\mathcal{A}$ with joining SU

Peering and clustering phase

SU joins and discovers the p2p network through $\mathcal{A}$
SU runs TwoWayEPID() with each peer
SUs of the overlay network form clusters $\{\mathcal{C}^{(i)}\}_{1 \le i \le n_{\mathcal{C}}}$
SUs $\in \mathcal{C}^{(i)}$ elect a leader $SU_L^{(i)}$, $\forall\ 1 \le i \le n_{\mathcal{C}}$
SUs $\in \mathcal{C}^{(i)}$ maintain a local blockchain $\mathcal{BC}^{(i)}$
SUs $\in \mathcal{C}^{(i)}$ run steps 2-6 of Rekeying($\mathcal{C}^{(i)}$) (Alg. 5)

Peering with *DB*s

DBs form validators set $\mathcal{V}$
Global blockchain $\mathcal{BC}$ is created with validators $\in \mathcal{V}$
DBs $\in \mathcal{V}$ and FCC maintain full copy of $\mathcal{BC}$
for $i = 1, \cdots, n_{\mathcal{C}}$ **do**
  $SU_L^{(i)}$ authenticates with DBs using EPID
  $SU_L^{(i)}$ peers up with DBs and becomes a validator
  $SU_L^{(i)}$ submits $y^{(i)}$ to $\mathcal{BC}$
  $SU_L^{(i)}$ requests a beacon $\beta^{(i)}$ from a DB
  DB sends an EPID challenge m to $SU_L^{(i)}$
  $SU_L^{(i)}$:$(\Sigma_L, \mathcal{P}_L)$←EPID.Sign($sk_L$, $\mathcal{K}_{pk}$, m, $\mathcal{L}_{sig}$, $\mathcal{L}_{iss}$)
  DB verifies $(\Sigma_L, \mathcal{P}_L)$ with EPID.Verify()
  DB issues $\beta^{(i)}$ to $SU_L^{(i)}$ and submits it to $\mathcal{BC}$
  $SU_L^{(i)}$ selects SUs $\in \mathcal{C}^{(i)}$ into $\mathcal{R}^{(i)}$
  Every $\mathcal{T}_{\beta}$, SUs $\in \mathcal{R}^{(i)}$ transmit β_i for a duration d

distributed (t_i, n_i)-TBLS signatures. To do this, SUs of $\mathcal{C}^{(i)}$ need to run the distributed TBLS.DKG protocol, described in Section III-A, 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, $z_j^{(i)} = g^{x_j^{(i)}}$. Cluster User Public Keys will

represent SUs' pseudonyms within $\mathcal{C}^{(i)}$ and are also used to identify SUs' transactions in $\mathcal{BC}^{(i)}$.

To handle system-wise access revocations, *TrustSAS* requires that each of these Cluster User Public Keys is associated with the EPID signatures of the corresponding SU over some statement that is known by all cluster members. To achieve this, each SU signs its Cluster User Public Key itself, which is known to all SUs in the cluster, using EPID.Sign with its EPID Membership Private Key, sk_{SU} (Alg 5, 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 5, step 5). Then each SU will send the signatures, obtained in steps 4 and 5 of Alg 5, to the leader $SU_L^{(i)}$, which will collect all these signatures and include them in $\mathcal{BC}^{(i)}$. Later, when an SU is detected to be malicious, the leader will add SU's Cluster User Public Key along with its EPID signature to $\mathcal{L}_{sig}$ to revoke its access to the system.

Algorithm 5 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($sk_j, \mathcal{K}_{pk}, z_j^{(i)}, \mathcal{L}_{sig}, \mathcal{L}_{iss}$)
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 $\mathcal{BC}$

3) PEERING WITH *DB*s (Alg. 4, STEPS 17-30)

Since SUs' clusters are now established, *DB*s can join the p2p network. This is done by making clusters leaders authenticate with *DB*s through EPID and then peer up with them.

During this phase, a global blockchain $\mathcal{BC}$ is also created to keep track of the key system-wise events. Only *DB*s and clusters leaders can be validators in the global blockchain to validate and propose 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 *DB*s 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 protocol. 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 [45]. $\mathcal{T}_\beta$ is a system design parameter that could be adjusted according to system dynamics and how frequent SUs join the system. A leader needs to request this beacon from one of the *DB*s and can acquire it only if it successfully proves its legitimacy to *DB* through EPID as depicted in steps 24-28 of Alg.4. This is achieved by creating an EPID signatures of a challenge message m that *DB* has created for this purpose. If the EPID signatures is successfully verified, *DB* issues a beacon to $SU_L^{(i)}$ and submits the beacon to $\mathcal{BC}$ so that accessible by all entites in the system. $SU_L^{(i)}$ 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 *DB*s and the leader of $\mathcal{C}^{(i)}$ as long as EPID revocation lists are 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. 6, 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 clusters. Failure to detect any beacons could mean that either no cluster is nearby or all nearby clusters are not accepting new SUs. In either case, SU will start a new cluster centered at its location and will request a beacon from one of the *DB*s and will itself start accepting new members, as described in Alg. 4.

Algorithm 6 Join $\mathcal{C}^{(i)}$

1: SU scans control channel for beacons in $\mathcal{BC}$
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 up with SUs $\in \mathcal{C}^{(i)}$ and downloads $\mathcal{BC}^{(i)}$
8: SUs $\in \mathcal{C}^{(i)}$ run REKEYING() in the 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 Algorithm 4

When a nearby new SU detects a beacon, it invokes the two-way verification procedure, TwoWayEPID, 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. An SU can be a member of only one cluster at a time.

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 deny 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. The rekeying steps are shown in Alg. 5. If rekeying is needed, it will take 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 choose to perform rekeying when malicious SU s are detected and denied access to the system, or when faulty SUs are detected.

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. 7.

Algorithm 7 Private Spectrum Query

1: $SU_L^{(i)}$ express interest to query DBs
2: DBs send an EPID challenge m to $SU_L^{(i)}$
3: $SU_L^{(i)}$: EPID.SIGN(sk_L, $\mathcal{K}_{pk}$, m, $\mathcal{L}_{sig}$, $\mathcal{L}_{iss}$)
4: $SU_L^{(i)}$ requests other $\tau - 1$ SUs to EPID.SIGN m
5: $SU_L^{(i)}$ sends τ EPID signatures of m to DBs
6: DBs verify the τ signatures with EPID.VERIFY()
7: **if** any signature is not valid **or** signatures are not unique **then**
8: DB adds $SU_L^{(i)}$ to $\mathcal{L}_{sig}$; **break**
9: **if** SUs $\in \mathcal{C}^{(i)}$ experience timeout from $SU_L^{(i)}$ **then**
10: SUs $\in \mathcal{C}^{(i)} \setminus \{SU_L^{(i)}\}$ 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. 4
13: $SU_L^{(i)*}$ adds $SU_L^{(i)}$ to $\mathcal{L}_{sig}$ and becomes $SU_L^{(i)}$
14: go to Step 1
15: $SU_L^{(i)}$: $\boldsymbol{\mathcal{D}_q} \leftarrow$ BATCHPIR(DBs, ℓ, 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 the 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 cluster SU members, and a cluster leader will only query DBs in one of three scenarios: Period allocated for using some channel(s) expires, quality of currently assigned channels degrades, currently used channels are requested to be vacated (e.g., requested by primary user of the channel).

1) AUTHENTICATION AND PERMISSION

For a cluster to acquire spectrum availability from DBs, *TrustSAS* requires that the cluster has 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. This will also force SUs to join existing clusters instead of creating their own clusters and trying to enjoy spectrum resources just by themselves. Therefore, before the leader $SU_L^{(i)}$ of a cluster $\mathcal{C}^{(i)}$ queries DBs for spectrum availability information, it needs to show $\mathcal{C}^{(i)}$ meets this requirement. For that, $SU_L^{(i)}$ needs to provide τ EPID signatures created by different legitimate SUs over a challenge message m that DBs created for this purpose as depicted in steps 2-5 of Alg.7. This is to show in a verifiable way that $\mathcal{C}^{(i)}$ has the minimum number of SUs required, and to prevent $SU_L^{(i)}$ from trying to deceive the system by pretending to meet this requirement. Note that EPID prevents $SU_L^{(i)}$ from forging these τ signatures without being detected. *TrustSAS* will not require these τ EPID signatures later unless a change in the membership of $\mathcal{C}^{(i)}$ takes place. If the EPID verification of these τ signatures is successful, then $SU_L^{(i)}$ can proceed with querying DBs for available channels. Otherwise, DBs will label $SU_L^{(i)}$ as malicious and will add it to the revocation list, $\mathcal{L}_{sig}$. To ensure robustness to $SU_L^{(i)}$'s failures, a timeout parameter could be implemented beyond which if SUs within $\mathcal{C}^{(i)}$ do not receive spectrum availability information from $SU_L^{(i)}$, it would be labeled as malicious, added to the revocation list, $\mathcal{L}_{sig}$, and a new leader will be elected. The Rekeying procedure is then run within $\mathcal{C}^{(i)}$ without the malicious/revoked leader, and the new leader will request a new beacon for the cluster as in steps 24-30 of Alg.4

2) SPECTRUM QUERYING

To privately query DBs for spectrum availability for the current $\mathcal{T}_{epoch}$, $SU_L^{(i)}$ collects SUs' queries in $\mathcal{C}^{(i)}$ and batches them together, and then runs the information theoretic *PIR* protocol with DBs via BatchPIR() described in Section III-D. $SU_L^{(i)}$ then submits the query response, $\boldsymbol{\mathcal{D}_q}$, as a block $\mathcal{B}_{epoch}$ for inclusion in local blockchain $\mathcal{BC}^{(i)}$. 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 misbehaving by sharing altered availability information. A leader SU is not trusted to divide spectrum resources among members of the cluster, and hence, *TrustSAS* instead relies on the concept of smart contracts which include spectrum allocation rules that are signed and defined by spectrum databases. In fact, once $\mathcal{B}_{epoch}$ is validated by SUs, $SU_L^{(i)}$ will trigger the execution of the included 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 $\mathcal{C}^{(i)}$. The execution of

the smart contracts will result in the automatic assignment of spectrum resources in a way that follows the guidelines of the *SAS* while ensuring coexistence between *SU*s. This assignment will be valid for the duration of the $\mathcal{T}_{epoch}$ period.

The smart contract, just like any other transaction in the blockchain, cannot be modified once added to the ledger. Thus, it will always be executed in the same way as intended and will respond to the changes in the state of the cluster-level blockchain, such as the current number of *SU*s in the cluster, the channel of interest, etc. As the execution of the smart contract will yield the same result for all the *SU*s within a cluster, this will give all cluster members an idea about how spectrum must be assigned, and will set expectations on how each *SU* may behave.

D. NOTIFYING ABOUT SPECTRUM USAGE

Once spectrum resources are allocated among *SU*s, $SU_L^{(i)}$ will need to share with the *DB*s the spectrum 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 information regarding the received signal strength in the used and adjacent frequencies, received packet error rates and other common standard interference metrics for all *SU*s 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 *DB*s 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 $\mathcal{L}_{sig}$. These steps are summarized in Alg. 8.

Algorithm 8 Spectrum Usage Notification

1: $SU_L^{(i)}$ constructs a block $\mathcal{B}_i$ with intended usage information
2: $SU_L^{(i)}$ shares $\mathcal{B}_i$ with SUs $\in \mathcal{C}^{(i)}$ 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$(\mathcal{B}_i, \sigma_{\mathcal{B}_i}, y_i)$ with BFT
6: **if** $val == True$ **then**
7: $\quad \mathcal{B}_i$ is added to $\mathcal{BC}$
8: $\quad$ *DB*s update their records
9: **else**
10: $\quad$ *DB*s flag $SU_L^{(i)}$ as malicious
11: $\quad$ $SU_L^{(i)}$ is added to revocation list $\mathcal{L}_{sig}$ in $\mathcal{BC}$
12: $\quad$ *DB*s remove $\beta^{(i)}$ from list of valid beacons on $\mathcal{BC}$

E. REINTIALISATION PHASE

The blockchains in the system do not need to grow indefinitely and thus present a storage burden especially on *SU*s with limited storage capacity. In fact, the system can periodically undertake a periodic checkpointing [4] operation that consists of making all validators of a blockchain reach consensus on the state of the blockchain at a specific point in time and create a checkpoint attesting on the validity of the blockchain up to that point in time. This way, when validators in the blockchain, which include leader *SU*s and *DB*s in the global blockchain and *SU*s and leader *SU* for each cluster, receive $2f + 1$ identical checkpoint messages from the different members of the blockchain, they can mark the checkpoint and clear the blocks preceding it [4]. The checkpoint could be created by hashing the blocks leading to the current agreed-upon blockchain state [4]. This checkpoint could also be stored in the global blockchain after being signed by all members of the cluster. Only a few nodes now will have to store the whole blockchain and this will help alleviate the storage overhead especially on *SU*s.

F. DETECTING MISBEHAVING SUs

Once spectrum resources are assigned and divided among *SU*s of a cluster, some *SU*s may misbehave and not follow the spectrum assignment rules agreed upon within the cluster. For simplicity and for illustration purposes, in *TrustSAS*, we consider the case where a misbehaving *SU* uses a channel that is different from the one that was assigned to it. Such behavior could be motivated by the fact that current assigned channel experiences quality degradation or some increased interference levels. Spectrum usage monitoring happens at the cluster level at the locations belonging to the cluster and not system wise. As the number of possible locations, represented as cells in our system, should be smaller than the number of member *SU*s, due to the fact that *SAS* systems are highly dense by nature, it should be the case that there will always be a *SU* that could be picked as a spectrum misuse detector in a specific cell/location.

To cope with such misbehaving *SU*s, *TrustSAS* follows an approach inspired by the SpecGuard [46] spectrum misuse detection protocol as we discuss in the following. *TrustSAS* requires each *SU* within a cluster to embed a spectrum permit into its physical-layer signals using proper power control in the modulation phase (using QPSK for instance). A spectrum permit refers to a cryptographic authorization to use a specific channel in a particular region for a certain duration [46]. It is assigned to *SU*s by the cluster leader based on the outcome of the spectrum assignment agreement within the cluster. This permit could later be decoded and verified by other *SU*s in the cluster.

For this, we assume that time duration for using a channel, $\mathcal{T}_{epoch}$, is slotted into $\gamma \geq 1$ slots. Based on the outcome of the spectrum assignment, $SU_L^{(i)}$ will do the following for every *SU* j in the cluster. First, $SU_L^{(i)}$ chooses a random number $\lambda_{j,\gamma}$ of sufficient length, 160 bits for instance, and recursively computes $\lambda_{j,k} = h(\lambda_{j,k+1})\ \forall k \in [0, \gamma - 1]$, where h is a cryptographic hash function. Then it sends $\lambda_{j,\gamma}$ to the corresponding *SU* j who also recursively computes $\{\lambda_{j,0}, \cdots, \lambda_{j,\gamma-1}\}$, where $\lambda_{j,k}$ represents the spectrum permit

of *SU j* in slot k of the current $\mathcal{T}_{epoch}$. Every *SU j* has to keep transmitting the spectrum permit $\lambda_{j,k}$ in slot k ($\forall k \in [1, \gamma]$) during $\mathcal{T}_{epoch}$. *SU* embeds $\lambda_{j,k}$ into physical layer signals using proper power control in the modulation phase.

Spectrum misuse detection within a cluster could be activated according to some random schedule over some randomly selected *SU*s or whenever there is a complaint about severe interference. For this, *TrustSAS* requires $SU_L^{(i)}$ to periodically select some *SU*s from $\mathcal{C}^{(i)}$ to scan the channels allocated to the cluster to detect any misbehaving *SU*s. The leader shares with these *SU*s the channel index, the starting time of the corresponding transmission time duration and the hash value $\lambda_{j,0}$ of every *SU j* that is being monitored.

Each misuse detector *SU* attempts to detect the k^{th} spectrum permit, denoted as $\lambda'_{j,k}$ of a monitored *SU j*, from the physical-layer signals, in the demodulation phase, over the specified channel for every time slot $k \in [1, \gamma]$ of the current $\mathcal{T}_{epoch}$. Then, it compares $\lambda_{j,0}$ with $h^k(\lambda'_{j,k})$, where $h^\eta(x)$ denotes η successive applications of the hash function h to x. If $\lambda'_{j,k} = \lambda_{j,k}$, i.e. the permit is authentic, then the equality $\lambda_{j,0} = h^k(\lambda'_{j,k})$ should hold. Otherwise, the corresponding *SU j* is most likely misusing the spectrum and it will be added to $\mathcal{L}$.

Besides operating on other channels, malicious *SU*s may also try to perform other malicious activities such as preventing the cluster from reaching consensus or from completing a group signature over a specific statement. These malicious *SU*s can easily thanks to the underlying BFT and TBLS mechanisms and they will be also revoked access to the system, which complies with the *accountability* requirement (stated in Section I-A).

V. SECURITY ANALYSIS

A. THREAT MODEL

TrustSAS assumes that *DB*s 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 *SU*s' information and try to infer it from the messages they receive from *SU*s. *TrustSAS* also assumes that these *DB*s do not collude with other *SU*s. Despite the fact that the collusion between *DB*s is improbable due to the fact that these *DB*s are operated and managed by competing service providers with no/little incentive to collude and also due to regulatory enforcement from bodies such as FCC to protect users' data, our system tolerates collusion up to a certain limit. This is thanks to the underlying properties of multi-server information-theoretic PIRs which are well studied in these works. [11], [47], [48]. We refer to a *SU* that faithfully follows the protocol as *honest*; otherwise, it is referred to as *Byzantine*. A Byzantine *SU* may behave arbitrarily, for instance, by refusing to participate in the protocol, or by colluding with other *SU*s to coordinate an attack on the system. However, *TrustSAS* assumes that these Byzantine *SU*s do not collude with *DB*s. For each cluster $\mathcal{C}^{(i)}$, *TrustSAS* requires that at least t_i out of the n_i *SU*s participate in the signature, and that no more than $f_i = (n_i - t_i)$ *SU*s are Byzantine. These t_i *SU*s serve as witnesses for the cluster to make sure that the leader is not communicating compromised information. Further, it is assumed that a malicious *SU* can lie about the channel that it intends to use to mislead *DB* and other *SU*s within the same cluster.

B. SECURITY OBJECTIVES

Given the aforementioned threat model, *TrustSAS* aims to achieve the following security objectives:

- *Private Spectrum Availability Query: SU*s can query *DB*s *privately*, without having to reveal their operational information.
- *Private Spectrum Usage Notification: SU*s can notify *DB*s about their channel usage and transmission parameters *privately*.
- *Robustness to Failures:* All security guarantees are maintained, even when a system entity (*SU* or *DB*) fails or is compromised.
- *Immutable Public Log for Auditability:* A globally consistent, tamper-resistant public log is maintained, where each system event, once produced and logged, cannot be altered or deleted.
- *Anonymity and Membership Verifiability: SU*s' authenticity can be verified before the *SU*s are granted system access, and *SU*s cannot be identified at any stage of protocol execution.
- *Location Privacy Protection of SUs: SU*s' physical location information is kept private at all times from all *DB*s. That is, *DB*s do not have access to any location traces of *SU*s, and even if they did these traces cannot be linked to specific *SU*s.

C. SECURITY RESULTS

Corollary 1: TrustSAS achieves unforgeability and robustness of TBLS signatures against an adversary that can corrupt any $f_i < n_i/2$ *SU*s 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 $\mathcal{C}^{(i)}$.

Proof: TrustSAS uses BFT consensus in conjunction with a threshold signature TBLS. BFT consensus within a cluster $\mathcal{C}^{(i)}$ can withstand f_i Byzantine *SU*s, where f_i is the upper bound for BFT mechanisms satisfying $n_i \geq 3f_i+1$ [49]. Since we also require t_i-out-of-n_i *SU*s for signing with TBLS, therefore, at most $f_i = n_i - t_i$ can turn Byzantine. This means that $t_i + f_i \geq 3f_i + 1$, yielding the above result. ■

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 $f_i < n_i/3$.

Proof: Since *TrustSAS* combines BFT consensus with TBLS threshold signatures, then this corollary follows naturally from Corollary 1 and the proof of Corollary 2. ■

Corollary 4: TrustSAS guarantees information-theoretic private spectrum availability query to *SU*s against the coalition of up to π *DB*s.

Proof: This is guaranteed by the underlying π*-out-of-*ℓ Shamir secret sharing scheme used in the *PIR* protocol that *TrustSAS* adopts. ■

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 *DB*s and *SU*s alike.

Proof: This follows naturally from the fact that *TrustSAS*'s underlying primitives, namely BFT consensus, TBLS signature, and *PIR* are all robust against Byzantine failures. ■

Corollary 7: TrustSAS guarantees location privacy information protection to all *SU*s.

Proof: In the spectrum availability query phase, *DB*s will only learn that $SU_L^{(i)}$ is legitimate via EPID but not the identity of the records, associated with locations of *SU*s in $\mathcal{C}^{(i)}$, that $SU_L^{(i)}$ is querying thanks to the information-theoretic *PIR* that *TrustSAS* uses. When sharing spectrum usage information, *TrustSAS* uses the Cluster Public Key $y^{(i)}$ as a pseudonym for $\mathcal{C}^{(i)}$, which reveals no information about *SU*s in $\mathcal{C}^{(i)}$ and thus locations cannot be linked to any specific *SU*s. ■

VI. PERFORMANCE EVALUATION

In this section, we assess the effectiveness of *TrustSAS* by evaluating the performance of each its building block and algorithms. These performances are evaluated both analytically and empirically via either implementations or benchmarking of the underlying mathematical and cryptographic operations using MIRACL library [50]. Experiments are carried out on a testbed that we built on Geni platform [51] using percy++ library [52]. 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.

A. CRYPTOGRAPHIC OVERHEAD

1) DISTRIBUTED KEY GENERATION (DKG)

In *TrustSAS*, DKG is used within each cluster to distributedly generate the cluster public and user secret keys. Its execution requires performing a number of elliptic curve point multiplications that is proportional to the number of *SU*s within the cluster. Using the benchmarking results that we derived using the MIRACL library [50], we provide in Table 2 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 *SU*s, where n_i is again the number of *SU*s in cluster $\mathcal{C}^{(i)}$. 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 2. DKG Overhead within cluster $\mathcal{C}^{(i)}$.

Operation	Analytical Cost	Empirical Cost
SU's Computation	$\mathcal{O}(n_i) \cdot PM$	$1.05\ s$
Communication	$\mathcal{O}(n_i)$ messages	$\propto 1000$ messages

Variables: PM: cost of an elliptic curve point multiplication. $n_i = 1000\ SUs$.

2) THRESHOLD SIGNATURE (TBLS)

Table 3 provides the analytical and emirical cost of the different TBLS operations executed by *SU*s in $\mathcal{C}^{(i)}$. *SU*s repeatedly sign the consensus statement at each consensus 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 3. 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 signing *SU*s 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 as illustrated in Table 3. To have an idea about the running time of TBLS's different operations, we use dfinity's implementation of TBLS [53]. We measure the time it takes to generate one signature share, which is performed by each *SU*, and the time it takes for the leader to verify all of the t shares before combining them into one cluster signature, assuming $t = 1000$. Finally, we measure the time it takes to verify the cluster's signature.

TABLE 3. TBLS complexity within $\mathcal{C}^{(i)}$.

Operation	Analytical Cost	Empirical Cost
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: TPO is the cost of one tate pairing. $Expp$ and $Mulpp$ are the cost of a modular exponentiation and multiplication, respectively, over modulus p. $t_i = 1000$

3) ENHANCED PRIVACY ID (EPID)

TrustSAS relies on EPID to allow *SU*s to anonymously prove their legitimacy without revealing their true identities thanks to its underlying zero-knowledge proof mechanism. We evaluate the complexity of EPID by assessing the cost of running EPID.Sign(), invoked by *SU* members, and EPID.Verify(), invoked by any *TrustSAS* entity. Both analytical and empirical efficiencies of EPID are depicted in Table 4.

TABLE 4. 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

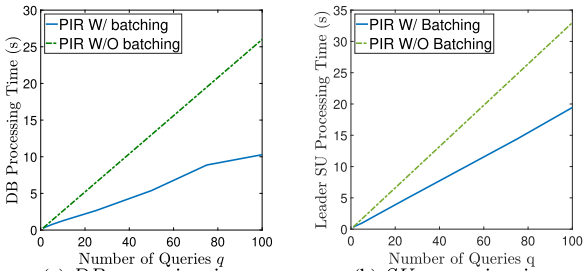
Variables: $\delta_1 = |\mathcal{L}_{priv}|$, $\delta_2 = |\mathcal{L}_{sig}|$, $\delta_3 = |\mathcal{L}_{iss}|$, and c is a constant. Cryptographic parameters correspond to 128-bit security level as recommended in [55].

Every EPID.Sign() operation requires *SU* to prove its knowledge of a membership private key and that its private key does not appear or was not used to create a signature in $\mathcal{L}_{sig}$ and $\mathcal{L}_{iss}$ revocation lists using zero knowledge proofs. These proofs require the signer to perform $6\delta_2 + 2\delta_3 + c$ modular exponentiations, where δ_2 and δ_3 are the sizes of $\mathcal{L}_{sig}$ and $\mathcal{L}_{iss}$, respectively, and c is a small constant [38]. As for the verification, every EPID.Verify() costs the verifier $\delta_1 + 6\delta_2 + 2\delta_3 + c$ modular exponentiations, where δ_1 is the size of $\mathcal{L}_{priv}$ revocation list. To translate these analytic costs into running time estimates, we use Intel's publicly available SDK [54]. These running time results are illustrated in Table 4.

Even though these delays seem relatively high, they are still reasonable in our system, especially that these membership proof operations are independent, unfrequent, and do not occur simultaneously, once the system setup is completed. Note that this proof of membership has a linear cost in the size of the revocation list and could 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 this threshold, FCC can create a new group and perform a rekeying operation. Each *SU* will then need to prove to FCC that it is a legitimate member and that its membership was not revoked, before acquiring a new membership private key for the new group from FCC. 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*)

TrustSAS adopts and relies on the information-theoretic, batching-enabled *PIR* proposed in [42] to enable private querying of *DB*s. We evaluate this protocol both empirically and analytically. For the empirical evaluation, we use percy++ [52] library deployed on a testbed that we built on Geni [51] cloud infrastructure. This testbed consists of 7 VMs, similar to the number of approved *SAS* operators, each playing the role of a *DB* and distributed over multiple locations. The Lenovo Yoga 3 Pro laptop used to play the role of the cluster leader. *TrustSAS* opted for using a batching-enabled *PIR*, which enables a cluster leader to lump together all queries, originated from the different *SU*s within its cluster, into one query instead of sending them separately to *DB*s. This reduces the overhead on both *DB*s and cluster leaders, as shown in Figs. 3a and 3b. In Table 5, we provide the analytic overhead expressions of the cluster leader computation, the *DB* query processing time, and the communication overhead.



(a) DB processing time. (b) SU processing time.

FIGURE 3. Overhead of *PIR* on *DB*s and cluster leaders.

TABLE 5. 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: ℓ: number of spectrum databases, q: number of batched PIR queries. DB size is η field $\mathbb{F}$ elements, $add_{\mathbb{F}}$ and $mul_{\mathbb{F}}$ denote the cost of an $\mathbb{F}$ addition and an $\mathbb{F}$ multiplication. Note in a field $\mathbb{F}$ with of characteristic 2, additions are equivalent to XOR and multiplications are equivalent to AND.
Experiment parameters: $\ell = 7, q = 25, \eta = 560\ MB$

B. BLOCKCHAIN OVERHEAD

This is another important component of *TrustSAS* that is invoked whenever there is a need to update the state of either the local or the global blockchains and ensures agreement among the different *TrustSAS* entities even in the presence of Byzantine behavior. It is worth reiterating that our system does not rely on public blockchains that require the notion of tokens or cryptocurrencies and use the computationally-intensive proof-of-work as a consensus mechanism, but it rather uses private blockchains where members perform byzantine fault-tolerant (BFT) consensus to come to an agreement about the state of the system.

For performance evaluation, we opt for the GoSig [56] protocol, one of the most efficient BFT protocols. We evaluate the communication overhead incurred by each *SU* in the cluster, which reflects the number of messages sent during every consensus round. As illustrated in Table 6, this number is quasi-linear in the size of the cluster, n_i, which translates into a total communication overhead of $\mathcal{O}(n_i^2 \log n_i)$. GoSig requires that all users participate in the consensus process. To have an estimate on how long it takes cluster leaders, *SU*s and *DB*s to reach a consensus over a specific block when using GoSig, we set the throughput between the nodes to 10 *Mbps* and the propagation delay among *SU*s to 20 *ms* (since *SU*s within a cluster are close to each other) and

TABLE 6. BFT complexity.

Operation	Analytical Cost	Empirical Cost	
		cluster	global
Messages per user	$\mathcal{O}(n_i \log n_i)$	$\propto 3000$	$\propto 26$
Consensus w/o failures	$\mathcal{O}(n_i^2 \log n_i)$	$4.3\ s$	$0.12\ s$
Consensus w/ failures	$\mathcal{O}(n_i^2 \log n_i)$	$6.32\ s$	$0.18\ s$

Parameters: number of users participating in BFT in cluster i: $n_i = 1000$, number of clusters leaders and DBs participating in global BFT: $n = 20$, bandwidth = $10Mbps$, 1 signature verification per SU.

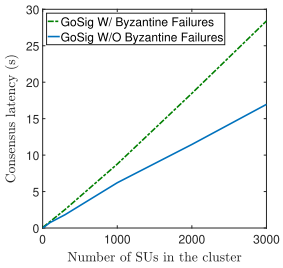


FIGURE 4. GoSig consensus latency.

simulate the protocol. Our results, depicted in Fig. 4, show that for a system with $n_{\mathcal{C}} = 15$ clusters and $\ell = 5$ DBs where clusters are of size as large as 1000 SUs, a consensus could be reached in less than 7 s even if up to $1/3$ of the SUs are Byzantine. This overhead is dominated by the cost of running BFT inside the clusters as the number of SUs within each cluster is expected to be larger than the number of clusters in the system as depicted in Table 6. The overhead of BFT in *TrustSAS* 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. 5. Note that the blockchain technology itself does not really incur a large computational burden as we are using private blockchains as opposed to public blockchains that rely mainly on proof-of-work which is computationally intensive. Most of the computational overhead in our system stems from running the REKEYING operation which involves verifying the signatures of all members of the cluster. However, REKEYING will run only occasionally, as membership changes are not frequent in the system, and most of the time, SUs will need to verify only a small number of signatures per consensus round, which will make the cost of running BFT similar to that depicted in Table 6.

An approach that can further reduce the cost in most of these algorithms is by not relying on all SUs in the BFT consensus, but rather use different quorums of users for every BFT round. This will reduce the overhead but will also have an impact on the security guarantees and robustness to failures.

The other main source of overhead in private blockchain systems such as ours stems from the relatively large communication overhead involved in the byzantine fault tolerant consensus that is required. We use a hierarchical approach that consists of dividing the system into two layers: a global layer where we use a blockchain that only cluster leaders and spectrum databases and a second layer consisting of multiple cluster specific blockchains visible only to members of each cluster. This layered approach saves tremendously in terms of communication overhead without losing in terms of security as we discussed earlier. Therefore, the use of blockchain technology should not present a computational burden on the system nor on the users' devices.

C. OVERALL SYSTEM OVERHEAD

1) END-TO-END DELAY OF *TrustSAS* COMPONENTS

We now estimate and provide the end-to-end delay achieved under *TrustSAS*. For this, we consider a $100km \times 100km$ region, divided into $100m \times 100m$ cells, and $\ell = 7$ spectrum databases whose content is of size $\eta = 560MB$ with $r = 10^6$ records each. SUs are grouped into 50 clusters each containing 1000 SUs, and τ is set to 10. *TrustSAS* entities communicate via Internet through 10 *Mbps* data-rate links. For simplicity, this evaluation assumes no Byzantine failures, and considers all of *TrustSAS*'s algorithms except Alg. 4, which is executed only at system setup. The cost of these algorithms is summarized in Table 7.

We start with the Rekeying procedure of Alg. 5. It first involves a DKG key generation operation between the 1000 SUs, which requires $1.05s$ of computation in total in addition to one round of reliable broadcast which takes $4.3s$ within the cluster. All members of the cluster need to perform TBLS and an EPID signature. As these two operations are done by each SU independently from other SUs in the cluster, performing them will only add $0.63ms$ for TBLS and 135 *ms* for EPID as illustrated in Tables 4 & 3. BFT consensus is then run to validate all public keys and signatures before adding them to the local blockchain, which takes around 72 s. In total, a Rekeying operation within a cluster of 1000 SUs takes around 77.47 s. We also evaluate the cost of a join event, depicted in Alg. 6, assuming that a beacon is detected. A join operation requires running TwoWayEPID, which involves 2 parallel EPID.Sign ($135ms$), 2 parallel EPID.Verify ($120ms$). It also requires a Rekeying operation (77.47 s) which brings the total delay for a join event to 78.12 s. Note that the relatively high cost of the Rekeying operation dominates the cost of a join operation, driving it to be high as well. However, this cost may still be reasonable when considering *TrustSAS* with unfrequent membership change events. Even in the case frequent membership change events, the Rekeying operation frequency can be set to be small enough to wait until enough join events take place before invoking the Rekeying operation to amortize the cost.

We also evaluate the end-to-end delay of running the private spectrum query depicted in Alg. 7. This involves

TABLE 7. End-to-end delay of *TrustSAS* algorithms.

Algorithm	Major Operations	Total Cost
Alg. 5 Rekeying within $\mathcal{C}^{(i)}$	DKG + TBLS.SignShareGen + EPID.Sign+ BFT(n_i)	77.47 s
Alg. 6: Join $\mathcal{C}^{(i)}$	TwoWayEPID + Rekeying	78.12 s
Alg. 7: Private Spec. Query	EPID.Sign + τ EPID.Verify + BatchPIR+ BFT(n_i)	13.15 s
Alg. 8: Spec. Usage Notif.	t_i (TBLS.SignShareGen + TBLS.SignShareVerif) + TBLS.SignReconstruct+ BFT($\ell + n_c$)	1.85 s

Experiment parameters: $n_i = 1000$, $t_i = n_i/2$, $\ell = 7$, $\tau = 10$, bandwidth = $10Mbps$. BFT(x) refers to one round of BFT among x parties.

τ parallel EPID.Sign operations with an overhead of 135 *ms*, τ EPID.Verify operations with an overhead of 1.2 *s*, one BatchPIR operation taking a total of 7.52 *s* for both $SU_L^{(i)}$ and *DB*s to process $q = 25$ *PIR* batched queries, and one round of BFT consensus within $\mathcal{C}^{(i)}$ taking 4.3 *s*. Running the smart contracts is done individually by each *SU* and does not require going through a consensus round. All giving a total of 13.15 *s* for running Alg. 7.

Finally, we evaluate the cost of the spectrum usage notification operation depicted in Alg. 8. This involves $t_i = n_i/2$ parallel TBLS.SignShareGen operations, taking 0.63 *ms*, t_i TBLS.SignShareVerif operations run by $SU_L^{(i)}$, taking 1.15 *s*, and one TBLS.SignReconstruct operation also run by $SU_L^{(i)}$, taking 461 *ms*. In total it takes $\mathcal{C}^{(i)}$ 1.61 *s* to prepare $\mathcal{B}_i$. Then, $\mathcal{B}_i$ goes through a round of BFT consensus between members of $\mathcal{V}$ containg $\ell = 7$ *DB*s and 50 clusters leaders to validate the block using TBLS.GroupSignVerif, requiring 237 *ms*. In total, the spectrum usage notification operation takes around 1.85 *s*.

Despite the relatively high cost of these algorithms, especially the rekeying and join operations, note that these operations are not frequent as we are not considering highly mobile *SU*s. Also the operations that are considered to be executed continuously and more frequently are the private spectrum queries and spectrum usage notifications, but even for these operations clusters are expected to query and notify *DB*s every few hours, as it is the case for TVWS systems which require *SU*s to query *DB*s every 24 hours.

2) SYSTEM LEVEL EVALUATION

To better understand the previously discussed results, in this section, we mimic a real system setup through simulation and assess the impact that *TrustSAS* might have on the operations of *SU*s. We run this simulation for 1000 $\mathcal{T}_{epoch}$s, which represent the lifetime of the system in this simulation.

For that, we model *SU*s' cluster join events during each $\mathcal{T}_{epoch}$ as a Poisson arrival process of rate λ (joining *SU*s per $\mathcal{T}_{epoch}$). *SU*s are distributed among the cells of the cluster following a two dimensional spatial Poisson process. We assume that each $\mathcal{T}_{epoch}$, set to 5 hours, is divided into small time slots each of duration $t = 15min$. Each *SU* could be either *active* or *inactive* during a time slot with its ON/OFF activity modelled as Bernoulli process with parameter p.

As discussed earlier, once the system is setup, join events are expected to be rare, and hence so do the Rekeying operations. We assess how frequent a cluster needs to run this operation for different *SU*s joining rates and we calculate the percentage of time the cluster needs to spend on this operation in the lifetime of the simulation. We set the maximum cluster size to 2000 *SU*s, beyond which the cluster will not accept any new members and therefore no rekeying will be needed. As depicted in Figure 5, this operation, despite its relatively high cost, will be run only for a small percentage of time of the system lifetime. When λ is small, Rekeying operations become frequent, and hence, it takes more $\mathcal{T}_{epoch}$s to reach the cluster capacity. As λ increases, the cluster capacity will be reached faster.

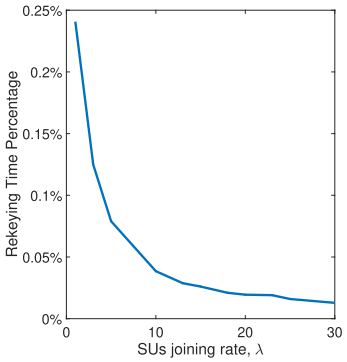


FIGURE 5. Percentage of time spent on Rekeying within a cluster.

We also study the impact of the activity of *SU*s within the cluster on the number of disruptions that every *SU* will experience. A disruption occurs due to a spectrum reassignment operation executed when the leader triggers the smart contracts on the local blockchain. A reassignment operation takes place if one or multiple *SU*s switch from an inactive state to an active state, and aims to adjust transmission parameters within the cluster to accommodate the newly active *SU*s. If *SU*s are active or inactive all the time, then they will not experience any disruption of their transmissions. A more reasonable scenario is that during a $\mathcal{T}_{epoch}$, every *SU* within the cluster could be active for some time slots and inactive for the remaining time slots. Here we assume that the leader has already acquired spectrum availability information for all the cells in its cluster. As depicted in Figure 6, as p increases, *SU*s experience more disruptions to their transmissions. This is due to the fact that higher values of p imply more contiguous active time slots for each *SU* which makes *SU*s more sensitive to any change of other *SU*s' states, i.e. from inactive to active.

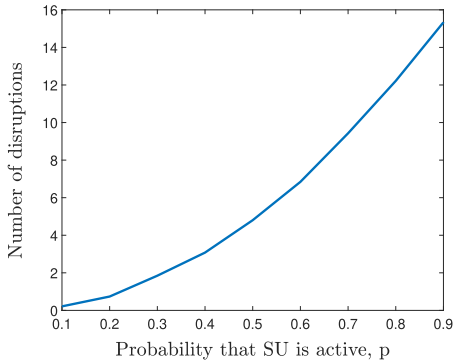


FIGURE 6. Average number of disruptions experienced by a *SU* depending on *p*.

D. COMPARISON WITH EXISTING APPROACHES

In this section we aim to compare *TrustSAS* to the two approaches that we believe are the closest to our work which are *P2-SAS* [19] and *PSEO* [5].

P2-SAS [19] relies on secure multi-party computation (SMPC) via Paillier partial homomorphic encryption. This severely limits the type of functions that this scheme can compute over the encrypted operational data of both *SU*s and *PU*s to only some limited/basic operations which may not work with the complex models used in SAS to calculate spectrum availability information. The use of SMPC may not be practical in SAS, since *PU*s are military and governmental entities that do not want to continuously engage in an interactive protocol with *SU*s and the spectrum database. Another limitation of SMPC solutions is that they are vulnerable to collusion among multiple parties. On top of these limitations, *P2-SAS* necessitates on average 6.96 seconds [19] to process a single spectrum query from a single *SU*, which means it would take *P2-SAS* 1.93 *h* to process requests from 1000 *SU*s. Our proposed protocol *TrustSAS*, on the other hand, needs only 13.15 *s* to handle a batched spectrum query of all 1000 *SU*s of a cluster in our system.

PSEO [5], on the other hand, requires *DB* to share an attenuation map of the whole covered region with all *SU*s to be used it to calculate the interference that *SU*s may cause to a *PU*. As discussed in Section I-C, this scheme requires a very large number of costly Paillier encryption over a large input size. Hence, it cannot scale with large numbers of *SU*s, expected to exist in large areas, especially given the fact that *SAS* is designed for highly dense areas [21], [22]. It also places a high trust on *SU*s by assuming that they will accurately and honestly calculate the interference. This overwhelms *SU*s with heavy computation, and reduces the role of *DB*s to just a simple gateway between *PU*s and *SU*s. Finally, *PSEO* requires that *SU*s communicate directly with *PU*s, which is not a realistic assumption as *PU*s in *SAS* systems are military and governmental entities. In terms of overhead, it takes *PSEO* 9.61 *s* to process a single *SU* request, or 2.67 *h* to process requests from 1000 *SU*s [5].

It is therefore clear that it takes existing approaches hours to perform what *TrustSAS* achieves in few seconds. It is also worthwhile noting that these aforementioned approaches consider a single-service provider *SAS* architecture, which deviates from the real-world architecture proposed by the FCC. Moreover, various *SAS* specific requirements were not considered. Specifically, they focus on the spectrum sharing between *PU*s and *SU*s and do not consider the coexistence between *SU*s themselves, the spectrum usage notification and system auditability, all of which are requirements specifically introduced for *SAS* by the FCC.

In summary, the main benefits that *TrustSAS* brings compared to alternative techniques are:

- Compliance with FCC's *SAS* requirements.
- Higher privacy and security guarantees achieved via existing privacy enhancing technologies.
- Is not concerned only with the privacy of SUs but also of all the entities involved in the system.
- Does not require *PU*s' involvement as done by some existing approaches, which use interactive secure multi-party computation that requires *PU*s' participation. This may not be practical in these systems since *PU*s are usually military and governmental entities.

However, there are still some limitations that require further investigation:

- Achieving higher privacy and security performances comes always at the cost of increased overhead. However, most of the overhead is incurred during the initialization phase or membership change events, which are expected to be infrequent in these studied systems.
- The storage overhead may also get relatively high, but this could be easily alleviated by performing a periodic re-initialization operation as discussed in Section IV-E.

We reiterate that this framework strikes a good balance between achieving high privacy guarantees to all system entities while complying with FCC's requirements and incurring an overhead that outweighs these benefits.

VII. CONCLUSION

In this paper, we propose *TrustSAS*, a trustworthy framework for *SAS* that preserves *SU*s' operational privacy while adhering to regulatory requirements mandated by FCC in the 3.5 GHz CBRS band. To realize this, *TrustSAS* synergizes state-of-the-art cryptographic mechanisms with the blockchain technology in an innovative way. We have shown the privacy benefits that *TrustSAS* brings to *SAS* environments through security analysis. We have also studied the performance of the proposed framework through theoretical analysis, simulations, and experimentation.

The preliminary results that we obtained in this paper are very promising, and show that it is possible to implement a *SAS* system that satisfies the FCC requirements while preserving the users privacy but still holding these users accountable if they misbehave. All is achieved while keeping a reasonable overhead that could be further optimized, as discussed in this paper, knowing that most of this overhead stems from operations that we believe will not be very frequent in

this kind of system. In light of this work and the results that we obtained, we highly recommend that the FCC considers some, if not all, of the proposed techniques to enhance security and privacy in *SAS* systems. In fact, we strongly believe that *SAS* systems are a natural application of multi-server *PIR* due to the fact that these systems satisfy the main requirement of multi-server *PIR*, the existence of multiple replicas of the spectrum database as mandated by the FCC. This will help provide location privacy to the users and encourage them to rely on this technology. The use of blockchains, as we demonstrated in this work, could also bring additional benefits especially in terms of auditability and accountability and also a platform for enforcing fair allocation of the spectrum through the use of smart contracts. FCC could also rely on EPID-like mechanisms to anonymously authenticate users to preserve their privacy while at the same time preserving the capability of evicting them from the system when they misbehave.

REFERENCES

[1] *Report and Order and Second Further Notice of Proposed Rulemaking*, document FCC 15-47, Federal Communications Commission, 2015.
[2] *Order on Reconsideration and Second Report and Order*, document FCC 16-55, Federal Communications Commission, 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, Apr. 2016.
[4] G. D. H. Hunt and L. Koved, "Blockchain checkpoints and certified checkpoints," U.S. Patent 10 586 210, Mar. 10, 2020.
[5] Q. Cheng, D. N. Nguyen, E. Dutkiewicz, and M. Mueck, "Preserving honest/dishonest users' operational privacy with blind interference calculation in spectrum sharing system," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2874–2890, Dec. 2020.
[6] V. Chen, S. Das, L. Zhu, J. Malyar, and P. McCann, *Protocol to Access White-Space (PAWS) Databases*, document RFC 6953, 2015.
[7] M. A. Clark and K. Psounis, "Trading utility for privacy in shared spectrum access systems," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 259–273, Feb. 2018.
[8] P. Marshall, *Three-Tier Shared Spectrum, Shared Infrastructure, and a Path to 5G*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
[9] *CBRS Communications Security Technical Specification*, document WINNF-15-s-0065, Wireless Innovation Forum, Apr. 2017.
[10] *CBRS Threat Model Technical Report*, document WINNF-15-p-0089, Wireless Innovation Forum, May 2016.
[11] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "When the hammer meets the nail: Multi-server PIR for database-driven CRN with location privacy assurance," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2017, pp. 1–9.
[12] M. Grissa, B. Hamdaoui, and A. A. Yavuz, "Location privacy in cognitive radio networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1726–1760, 3rd Quart., 2017.
[13] Z. Gao, H. Zhu, Y. Liu, M. Li, and Z. Cao, "Location privacy in database-driven cognitive radio networks: Attacks and countermeasures," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2751–2759.
[14] C. Guan, A. Mohaisen, Z. Sun, L. Su, K. Ren, and Y. Yang, "When smart TV meets CRN: Privacy-preserving fine-grained spectrum access," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1105–1115.
[15] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "Cuckoo filter-based location-privacy preservation in database-driven cognitive radio networks," in *Proc. World Symp. Comput. Netw. Inf. Secur. (WSCNIS)*, Sep. 2015, pp. 1–7.
[16] M. Grissa, A. Yavuz, and B. Hamdaoui, "An efficient technique for protecting location privacy of cooperative spectrum sensing users," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 915–920.
[17] M. Clark and K. Psounis, "Achievable privacy-performance tradeoffs for spectrum sharing with a sensing infrastructure," in *Proc. 14th Annu. Conf. Wireless On-Demand Netw. Syst. Services (WONS)*, Feb. 2018, pp. 103–110.
[18] S. Bhattarai, P. R. Vaka, and J.-M. Park, "Thwarting location inference attacks in database-driven spectrum sharing," *IEEE Trans. Cognit. Commun. Netw.*, vol. 4, no. 2, pp. 314–327, Jun. 2018.
[19] Y. Dou, K. C. Zeng, H. Li, Y. Yang, B. Gao, C. Guan, K. Ren, and S. Li, "P^2-SAS: Preserving users' privacy in centralized dynamic spectrum access systems," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 321–330.
[20] Q. Cheng, D. N. Nguyen, E. Dutkiewicz, and M. D. Mueck, "Protecting operational information of incumbent and secondary users in FCC spectrum access system," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.
[21] *Google Spectrum Access System*. Accessed: Sep. 11, 2019. [Online]. Available: https://www.google.com/get/spectrumdatabase/
[22] M. D. Mueck, S. Srikanteswara, and B. Badic, "Spectrum sharing: Licensed shared access (LSA) and spectrum access system (SAS)," Intel, Portland, OR, USA, White Paper, 2015.
[23] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "TrustSAS: A trustworthy spectrum access system for the 3.5 GHz CBRS band," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1495–1503.
[24] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 51–68.
[25] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," 2018, *arXiv:1805.04548*. [Online]. Available: http://arxiv.org/abs/1805.04548
[26] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2018, pp. 66–98.
[27] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Oct. 1995, pp. 41–50.
[28] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *Proc. Int. Workshop Public Key Cryptogr.* Berlin, Germany: Springer, 2003, pp. 31–46.
[29] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, Sep. 2004.
[30] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996.
[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1999, pp. 295–310.
[32] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[33] S. Goldfeder, J. Bonneau, R. Gennaro, and A. Narayanan, "Escrow protocols for cryptocurrencies: How to buy physical goods using bitcoin," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 321–339.
[34] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*. Pseudonym in Bitcoins, 2008.
[35] M. Vukolić, "Rethinking permissioned blockchains," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts*, Apr. 2017, pp. 3–7.
[36] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Secur.* Cham, Switzerland: Springer, 2015, pp. 112–125.
[37] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[38] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 3, pp. 345–360, May 2012.
[39] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. 11th ACM Conf. Comput. Commun. Secur. (CCS)*, 2004, pp. 132–145.
[40] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 1991, pp. 433–444.
[41] I. Goldberg, "Improving the robustness of private information retrieval," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 131–148.

[42] W. Lueks and I. Goldberg, "Sublinear scaling for multi-client private information retrieval," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2015, pp. 168–186.
[43] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proc. 36th Annu. ACM Symp. Theory Comput. (STOC)*, 2004, pp. 262–271.
[44] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *Proc. NDSS*, vol. 11, 2011, pp. 1–17.
[45] T. Chen, H. Zhang, G. M. Maggio, and I. Chlamtac, "CogMesh: A cluster-based cognitive radio network," in *Proc. 2nd IEEE Int. Symp. New Frontiers Dyn. Spectr. Access Netw. (DySPAN)*, Apr. 2007, pp. 168–178.
[46] X. Jin, J. Sun, R. Zhang, Y. Zhang, and C. Zhang, "SpecGuard: Spectrum misuse detection in dynamic spectrum access systems," *IEEE Trans. Mobile Comput.*, vol. 17, no. 12, pp. 2925–2938, Dec. 2018.
[47] M. Grissa, B. Hamdaoui, and A. A. Yavuz, "Unleashing the power of multi-server PIR for enabling private access to spectrum databases," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 171–177, Dec. 2018.
[48] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "Location privacy in cognitive radios with multi-server private information retrieval," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 4, pp. 949–962, Dec. 2019.
[49] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OSDI*, vol. 99, 1999, pp. 173–186.
[50] Multiprecision Integer and C Rational Arithmetic. (2013). *C++ Library (Miracl)*. Accessed: Jun. 2, 2018. [Online]. Available: https://github.com/miracl/MIRACL
[51] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Comput. Netw.*, vol. 61, pp. 5–23, Mar. 2014.
[52] *Percy++ Library*. Accessed: Jun. 14, 2018. [Online]. Available: http://percy.sourceforge.net
[53] *Threshold BLS Dfinity Implementation*. Accessed: Jun. 2, 2018. [Online]. Available: https://github.com/dfinity/random-beacon
[54] *The Intel(r) Enhanced Privacy Id Software Development Kit*. Accessed: Jun. 2, 2018. [Online]. Available: https://github.com/Intel-EPID-SDK
[55] *BlueKrypt—V32.3, Damien Giry, Cryptographic Key Recommendation*. Accessed: Jun. 2, 2018. [Online]. Available: https://www.keylength.com/
[56] P. Li, G. Wang, X. Chen, and W. Xu, "Gosig: Scalable Byzantine consensus on adversarial wide area network for blockchains," 2018, *arXiv:1802.01315*. [Online]. Available: http://arxiv.org/abs/1802.01315

MOHAMED GRISSA (Student Member, IEEE) received the Diploma of Engineering degree (Hons.) in telecommunication engineering from the Ecole Superieure des Communications de Tunis (Sup'Com), Tunis, Tunisia, in 2011, and the M.S. and Ph.D. degrees in electrical and computer engineering (ECE) from Oregon State University, Corvallis, OR, USA, in June 2015 and September 2018, respectively.

Before joining Oregon State University, he worked as a Value Added Services Engineer with the Orange France Telecom Group, from 2012 to 2013. His research interests include privacy and security in computer networks, cognitive radio networks, spectrum access systems, the IoT, Blockchain, and eHealth systems.

ATTILA ALTAY YAVUZ (Member, IEEE) received the M.S. degree in computer science from Bogazici University, Istanbul, Turkey, in 2006, and the Ph.D. degree in computer science from North Carolina State University, in August 2011. He was a member of the Security and Privacy Research Group with the Robert Bosch Research and Technology Center North America, from 2011 to 2014. He was an Assistant Professor with the School of Electrical Engineering and Computer Science, Oregon State University, from August 2014 to August 2018. He is currently the Director of the Applied Cryptography Research Laboratory and the Co-Director of the Center for Cryptographic Research with the University of South Florida (USF). He has authored 79 products, including research articles in top conferences, journals, and patents. His research on privacy-enhancing technologies and intra-vehicular network security is in the process of technology transfer with potential worldwide deployments. He is broadly interested in the design, analysis, and application of cryptographic tools and protocols to enhance the security of computer networks and systems. He is a member of ACM. He was a recipient of the NSF CAREER Award, the Cisco Research Award (twice), the USF Faculty Outstanding Research Achievement Award, and the USF College of Engineering's Outstanding Research Achievement Award.

BECHIR HAMDAOUI (Senior Member, IEEE) received the M.S., C.S., and Ph.D. degrees in electronics and communications engineering (ECE) from the University of Wisconsin-Madison, in 2002, 2004, and 2005, respectively.

He is currently a Professor with the School of Electrical Engineering and Computer Science, Oregon State University. His research interests include computer networking, network security, and wireless communication, with a current focus on edge cloud computing, network analytics, autonomous systems, dynamic spectrum management, 5G systems, and datacenters. He is a Senior Member of the IEEE Computer Society, the IEEE Communications Society, and the IEEE Vehicular Technology Society. He won several awards, including the ISSIP 2020 Distinguished Recognition Award, the ICC 2017 Best Paper Award, the 2016 EECS Outstanding Research Award, and the 2009 NSF CAREER Award. He also chaired/co-chaired many IEEE conference programs/symposia, including the 2017 INFOCOM Demo/Posters program, the 2016 IEEE GLOBECOM Mobile and Wireless Networks symposium, and many others. He served as a Distinguished Lecturer for the IEEE Communication Society for 2016 and 2017. He currently serves as the Chair for the IEEE Communications Society's Wireless Technical Committee (WTC). He serves/served as an Associate Editor for several journals, including IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE NETWORK, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

CHITTIBABU TIRUPATHI received the B.Tech. degree in computer science engineering from AP IIIT, India, in 2015. He is currently pursuing the M.Sc. degree from the School of Electrical Engineering and Computer Science (EECS), Oregon State University, Corvallis, OR, USA. Before pursuing masters, he worked as an Assistant System Engineer with Tata Consultancy Services Ltd., Bengaluru, India, from 2015 to 2017. His research interests include computer networks, network security, and applying machine learning techniques to IoT devices to make them intelligent enough to be more useful in the real world.