zkSAS: Practical Zero-Knowledge Proofs for Verifiable Spectrum Access Management

Nishat F. Purbasha*, Ifteher Alom*, Eric W. Burger†, Y. Thomas Hou†, Wenjing Lou†, Yang Xiao*
*University of Kentucky, Lexington, KY, USA
†Virginia Tech, Blacksburg, VA, USA

***Abstract*—Dynamic Spectrum Access (DSA) through the Spectrum Access Systems (SAS) elevates spectral efficiency, yet existing centralized models face allocation logic opaqueness and a lack of independent verifiability. While blockchain-based SAS architectures offer transparency and verifiability by default, they introduce critical privacy risks and prohibitive on-chain computational overhead. We introduce zkSAS, a practical zero-knowledge proof (ZKP) system designed to address the verifiability and privacy gaps in SAS deployments, with direct applicability to both the existing CBRS SAS model and blockchain-based SAS models. The system features a suite of ZKP circuits, encompassing proofs of allocation constraint validity and proofs of move list validity to verify that channel assignments and move list-based incumbent protection measures, respectively, adhere to regulatory constraints without exposing sensitive user data. Comprehensive evaluation of our prototype in both centralized and blockchain-based settings indicates that while proof generation scales with spectrum user population, verification remains lightweight and constant-time. We envision that zkSAS offers a scalable and practical path to secure, verifiable dynamic spectrum sharing.**

***Index Terms*—Spectrum access system, zero-knowledge proof, privacy, verifiability, policy compliance.**

I. Introduction

Dynamic spectrum access (DSA) has emerged as a transformative paradigm for optimizing spectral efficiency by opening frequency bands that are traditionally reserved for exclusive incumbent use to opportunistic access by commercial secondary users. As a prominent realization of this concept, the Spectrum Access System (SAS) [1] designed for serving the 3.55-3.7 GHz Citizens Broadband Radio Service (CBRS) band, has seen widespread adoption since 2020. Architecturally, an SAS functions as a centralized coordinator and database, dynamically authorizing spectrum grants for commercial users while protecting the preemptive access rights of incumbents such as naval radars and satellite ground stations.

However, current SAS implementations face an outstanding verifiability challenge, which stems from the centralized service model characterized by opaque allocation logic. Since the spectrum access right is an intangible asset, its value depends entirely on the validity and enforceability of the grants created by the SAS. Without a mechanism for independent verifiability, secondary users must place blind trust in the SAS for executing complex interference calculations and policy compliance. This lack of transparency and verifiability creates significant legal and commercial risks, as erroneous allocations can lead to destructive interference to incumbent and priority access users or the unfair exclusion of secondary access users. Such faults are difficult to audit in a closed-server architecture. Therefore, establishing the verifiability of SAS operations is essential for the reliability of a spectrum economy where access rights can be trusted and legally defended.

Meanwhile, recent research has explored decentralized, blockchain-based SAS architectures that realize verifiable spectrum management [2]–[7]. A blockchain system offers a unique advantage primarily due to its built-in verifiability, transparency, irreversibility, and limited trust assumptions on central service providers [2]. While varying in implementations, existing blockchain-based SAS solutions typically employ "on-chain" programs, commonly known as smart contracts, to execute channel assignment logic. Some proposals also incorporate on-chain compensation logic and trading mechanisms for short-term band leasing [3], [6], echoing the FCC vision that the blockchain technology has the potential to enable a dynamic spectrum economy with light administrative overhead [8].

Despite filling the verifiability gap, blockchain-enabled SAS models inevitably bring new challenges in privacy and computational efficiency. First, the inherent transparency of blockchain ledgers exposes sensitive operational data, including user device identifier, geo-location, temporal access pattern, etc., to all blockchain observers. The exposed user records also increase the risk of movement tracking on incumbent users. Meanwhile, spectrum allocation algorithms are often considered as intellectual property of SAS service providers; encoding them on-chain erodes business interests. Second, on-chain spectrum assignment imposes a heavy computational overhead due to the replicated execution model defined by blockchain consensus. Such overhead significantly restricts the complexity of the spectrum allocation logic. Existing algorithms such as the standardized Iterative Allocation Process (IAP) [9] and resource optimization-based allocation methods [10]–[14] would incur prohibitive on-chain costs.

Contributions. We introduce **zkSAS**, a practical zero-knowledge proof (ZKP) system that fills the critical verifiability gap in SAS deployments. zkSAS can be used independently to elevate the trustworthiness of existing SAS models by proving spectrum allocation validity to clients and facilitating regulatory audits. zkSAS can also be integrated into blockchain-based SAS to bridge the privacy-efficiency gap by generating off-chain execution proofs. These proofs certify that spectrum allocations produced in an off-chain private environment are valid and serve as direct triggers for any on-

chain logic contingent on a verified allocation.

Concretely, the zkSAS system at its core consists of a suite of ZKP constructions designed to prove the validity of two classes of spectrum operational outcomes.

- **Proof of allocation constraint validity.** We construct ZKP circuits that verify whether the channel assignments in a set of grants satisfy a comprehensive list of constraints established in prior work [1], [14]. To ensure both privacy and authenticity of allocation parameters binding to the proof, we employ a commit-and-prove design in which the specific allocation parameters are treated as a private witness while their cryptographic commitment is exposed as a public signal. This design ensures that users receive the authentic allocation associated with the proof and enables independent audits by a third party, without exposing sensitive user-specific assignment data. In our implementation, while proving time grows with the total number of counties and users per county encoded in the allocation, verification remains lightweight and nearly constant-time. (Section III)
- **Proof of move list validity.** We construct ZKPs for Move List [15], an essential function that protects incumbents by determining which commercial users should stop transmitting. Our proof construction encompasses the validity of key algorithmic checks rather than encoding the entire move list procedure in-circuit. Complex steps such as sorting and interference aggregation are executed off-chain, and their outputs are supplied as private witnesses. Similar to the proof of allocation constraint validity, while proving time grows with the size of the set of grants encoded in the witness, verification remains constant-time. (Section IV)

We provide three use cases of zkSAS in the contexts of the existing CBRS SAS paradigm and blockchain-based SAS models, followed by a new use case involving Spectrum Bux for future spectrum sharing (Section V). We implemented a zkSAS prototype and evaluated its computational overhead in both centralized SAS and blockchain-based SAS settings (Section VI). Across all settings, the dominant overhead comes from proof generation, whereas the verification is done efficiently and largely insensitive to the spectrum user population. Overall, the measured end-to-end latency stays practical at realistic county- and user-scale workloads, and it scales in a predictable way as the user population increases.

II. Preliminaries

A. Spectrum Access System Basics

SAS service model. Existing SAS implementations for the CBRS band follow a centralized client-server architecture. A spectrum user must register its devices, formally known as CBRS Devices (CBSDs), with a certified SAS administrator, accompanied by critical operational parameters such as geographic coordinates, antenna height, and device category [1]. Following registration, the user performs an `Inquiry` to identify available channels in its vicinity. To utilize a specific channel, the user then submits a `Grant` request; the SAS evaluates this request against the current interference environment and incumbent protection requirements, which is typically executed by a spectrum allocation algorithm (see below). Once a grant is issued, the device remains silent until it begins to periodically send out `Heartbeat` messages to the SAS. This periodic signaling mechanism allows the SAS to dynamically manage the spectral environment and authorize the CBSD for radio transmission at a short time scale.

Allocation algorithms. The generation of grants typically involves an allocation algorithm. The standard baseline is the Iterative Allocation Process (IAP), which iteratively evaluates the feasibility of each grant request by calculating aggregate interference and ensuring it remains below defined protection thresholds. While SAS administrators are permitted to use their proprietary allocation algorithms, the allocation result should be substantively similar to IAP. In current commercial deployment, allocation is performed during the nightly synchronization windows mandated by the Coordinated Periodic Activities among SASs (CPAS) procedure [9].

Incumbent protection with move list. A vital mission of SAS is to protect the pre-emptive access rights of incumbents. An incumbent, such as a naval radar, can arrive at any location in one of the Dynamic Protection Areas (DPAs). A SAS detects incumbent arrivals at DPAs via its Environmental Sensing Capability (ESC) units, which are deployed at fixed DPA points that reside at the border of DPAs. The SAS is required by law to suspend commercial grants, and signal affected users to stop transmitting in incumbent-occupied channels within five minutes [15]. To facilitate the determination of grant suspension, the *move list* algorithm is standardized [16] following the principle that higher interference creators are first to be put into the move list and have their grants suspended; the remaining grants are not affected.

B. List of Spectrum Allocation Constraints

A spectrum allocation specifies the channel assignments to users by an SAS. We summarize the spectrum assignment notations and allocation constraints according to Jai et al. [14]. Formally, an *allocation* is defined as a set of channel assignments to all registered users (extracted from the grants):

$$\mathbf{alloc} := \left\{ \left\{ x^c_{i|j} \right\}_{i \in \mathcal{P}_j, c \in \mathcal{F}_\mathrm{P}}, \left\{ y^c_{i|j} \right\}_{i \in \mathcal{G}_j, c \in \mathcal{F}} \right\}_{j \in \mathcal{A}}$$

where $x^c_{i|j} \in \{0,1\}$ is the binary indicator for PAL channel assignment—it equals 1 if channel c is assigned to PAL user i for county j. Similarly, $y^c_{i|j} \in \{0,1\}$ indicates whether channel c is assigned to GAA user i for county j. $\mathcal{P}_j$ is the set of PAL users in county j; $\mathcal{G}_j$ is the set of GAA users in county j. $\mathcal{F}_\mathrm{P}$ is the set of PAL-eligible channels (i.e., channels 1 to 10 in the CBRS band); $\mathcal{F}$ is the set of all channels. Lastly, $\mathcal{A}$ is the set of counties considered in this allocation.

***Constraint** 1 (PAL Assignment per Channel-and-County):* At most one PAL user can be assigned to each PAL-eligible channel in any county. That is,

$$\forall c \in \mathcal{F}_\mathrm{P}, \forall j \in \mathcal{A}: \qquad \sum_{i \in \mathcal{P}_j} x^c_{i|j} \leq 1 \tag{1}$$

***Constraint** 2 (PAL Assignment per County):* The number of channels assigned to any PAL user in any county equals the number of channel licenses held by that PAL user for that county. That is,

$$\forall i \in \mathcal{P}_j, \forall j \in \mathcal{A}: \qquad \sum_{c \in \mathcal{F}_\mathrm{P}} x^c_{i|j} = L^P_{i|j} \tag{2}$$

where $L^P_{i|j}$ is the number of licenses that PAL user i holds in county j.

***Constraint** 3 (PAL Interference Protection):* The aggregate interference received by a PAL user device on any channel, which is contributed by other PAL users and GAA users, should not exceed a threshold. That is,

$$\begin{aligned} &\forall c \in \mathcal{F}_\mathrm{P}, \forall l \in \mathcal{A}, \forall k \in \mathcal{P}_l, \forall a \in \mathcal{B}_{k|l}: \\ &\qquad x_{k|l} \sum_{j \in \mathcal{A}, j \neq l} \sum_{i \in \mathcal{P}_j, i \neq k} \sum_{b \in \mathcal{B}_{i|j}} x^c_{i|j} \cdot \gamma^c_{b,a} + \\ &\qquad\qquad x_{k|l} \sum_{j \in \mathcal{A}} \sum_{i \in \mathcal{G}_j} y^c_{i|j} \cdot \delta^c_{i,a} \leq T^\mathrm{P}_k \end{aligned} \tag{3}$$

where T^P_k is the interference threshold for PAL user k, which we assume, without impacting our later design, to be uniform across all channels and counties. $\mathcal{B}_{k|l}$ is the set of devices of PAL user k in county l; $\gamma^c_{b,a}$ is the received signal strength (RSS) from PAL user device b on channel c, evaluated at a location in the Priority Protection Area (PPA) of device a that is closest to b; $\delta_{i,a}$ is the RSS from GAA user i on channel c, evaluated at a location in the PPA of PAL user device a that is closest to i.

***Constraint** 4 (GAA Assignment):* The number of channels assigned to each GAA user in any county should not exceed a target number. That is,

$$\forall i \in \mathcal{G}_j, \forall j \in \mathcal{A}: \qquad \sum_{c \in \mathcal{F}} y^c_{i|j} \leq L^\mathrm{G}_{i|j} \tag{4}$$

where $L^\mathrm{G}_{i|j}$ is such target number for GAA user i in county j.

***Constraint** 5 (GAA Mutual Interference Control):* Each pair of GAA users assigned to any common channel should be geographically separated so that they do not interfere with each other's transmission. That is,

$$\begin{aligned} &\forall c \in \mathcal{F}, \forall j \in \mathcal{A}, \forall a, b \in \mathcal{G}_j, a \neq b: \\ &\qquad D_{a,b} \geq y^c_{a|j} \cdot y^c_{b|j} \cdot (R_a + R_b) \end{aligned} \tag{5}$$

where $D_{a,b}$ is the distance between GAA users a and b; R_a and R_b are their transmission ranges, respectively. In SAS operation, $D_{a,b}$, R_a, and R_b can be conveniently obtained from users' spectrum requests.

***Constraint** 6 (Incumbent Interference Protection):* If there is an active incumbent, the aggregated interference from all PAL and GAA users on any incumbent-active channel in any DPA should not exceed a threshold. That is,

$$\begin{aligned} &\forall c \in \mathcal{F}, \forall m \in \mathcal{I}: \\ &\qquad Z^c_m \sum_{j \in \mathcal{A}} \sum_{i \in \mathcal{P}_j} \sum_{b \in \mathcal{B}_{i|j}} x^c_{i|j} \cdot \alpha^c_{b,m} + \\ &\qquad\qquad Z^c_m \sum_{j \in \mathcal{A}} \sum_{i \in \mathcal{G}_j} y^c_{i|j} \cdot \beta^c_{i,m} \leq T^\mathrm{I}_m \end{aligned} \tag{6}$$

where T^I_m is the interference threshold for all locations in DPA m. Z^c_m is a binary indicator—it equals 1 if channel c is used by an active incumbent in DPA m. $\alpha^c_{b,m}$ is the RSS from PAL user device b on channel c, evaluated at a DPA point on DPA m's boundary that is closest to b; $\beta^c_{i,m}$ is the RSS from GAA user i on channel c, evaluated in a similar manner.

C. Zero-knowledge Proof

Zero-Knowledge Proof (ZKP) is a general protocol between two parties that allows the prover to convince the verifier that it knows a secret (or "witness") for which a given statement is true without revealing any information beyond the truth of the statement. Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARK) [17] is a type of ZKP system specifically designed for efficiently proving the validity of a computation. Theoretically, they rely on the Knowledge-of-Exponent assumption [18], which guarantees that any prover capable of producing a valid proof must computationally "know" the witness (not just that the witness exists). zk-SNARKs are characterized by their small proof size (*succinctness*), asynchronous interaction (*non-interactivity*), and sublinear verification time, regardless of the complexity of the underlying computation.

Technically, a zk-SNARK is constructed as a modular combination of two layers: an information-theoretic layer, which compiles a computation into an arithmetic circuit and attests to its satisfiability, and a cryptographic layer, which enforces integrity via a polynomial commitment scheme and achieves non-interactivity using the Fiat-Shamir transformation [19]. Together, a zk-SNARK allows a prover to prove knowledge of a witness w for a public statement x such that it satisfies the relation $C(x, w) = 0$ for some arithmetic circuit C, without revealing w (*zero-knowledge*). The system guarantees that an honest prover can always convince the verifier (*completeness*) and that a dishonest prover cannot convince an honest verifier that a false statement is true (*soundness*).

In our design, we use the Groth16 [20] implementation of zk-SNARK, mainly because of its verification time efficiency and availability of the supporting open-source JavaScript library `snarkjs` [21]. Groth16 achieves the theoretical lower bound for proof size (3 group elements) and verification time (3 pairings + n multiplications). This efficiency is achieved with the help of a circuit-specific trusted setup phase to generate the Common Reference String (CRS), which pre-computes a proving key linear in the circuit size and a constant-size verification key. We use Circom [22], a specialized domain-specific language for creating arithmetic circuits, to convert the spectrum allocation policy constraints into Rank-1 Constraint Systems (R1CS). R1CS is a system of quadratic constraints that represents an arithmetic circuit over a finite field and handles equality and comparison constraints differently. We will describe how we build R1CS constraints from the validity conditions in our proofs.

III. ZKPs for Allocation Constraint Validity

In this section, we construct a ZKP system for checking the validity of allocation constraints described in Section II-B.

A. ZKP Circuits for Allocation Constraints

Our general strategy is to build a verification circuit that arithmetizes all constraints while maintaining data confidentiality. For all constraints, we treat the primary variables as *private inputs* (i.e., the secret *witness* of the proof), including the channel assignment indicators ($x^c_{i|j}$, $y^c_{i|j}$, and Z^c_m), per-user channel assignment limits ($L^P_{i|j}$ and $L^G_{i|j}$), RSS values ($\gamma^c_{b,a}$ and $\delta i, a^c$), distance values ($D_{a,b}$, R_a, and R_b), and interference thresholds (T^{P}_k and T^{P}_k). This mitigates information leakage about the allocation to the verifier.

1) Constraints 1, 2, 4: Constructions for these constraints are straightforward as they only involve integer operations.

For Constraint 1, as shown in Eq. (1), the circuit enforces *booleanity* of each $x^c_{i|j}$ and an *at-most-one* condition per county, holding true if the summation equals 0 or 1. For Constraint 2, as shown in Eq. (2), the circuit enforces the *cardinality constraint* that the summation equals to $L^P_{i|j}$ together with explicit *range constraints* reflecting the system policy: $\forall i : 1 \leq L^P_{i|j} \leq 4$ and $\sum_{i\in\mathcal{P}_j} L^P_{i|j} \leq 7$. These checks ensure that each PAL holder selects exactly as many channels as their license count permits, and thus satisfy the county-wide licensing limit specified by the policy. For Constraint 4, as shown in Eq. (4), the circuit enforces booleanity of each $y^c_{n|j}$ and cardinality constraint that the summation equals $L^G_{i|j}$, together with the explicit range constraint reflecting the system policy: $\forall i : 0 \leq L^G_{i|j} \leq 4$. This guarantees that no GAA user exceeds their allowed target number of channel assignments.

2) Constraints 3, 5, 6: Constructions for these constraints are more complex as they involve handling fixed-point arithmetic and nested summations.

For Constraint 3, as shown in Eq. (3), the circuit computes the total interference as a sum of gated contributions using coefficients γ and δ, and multiplies by the local activity indicator $x^c_{k|l}$ so that the bound is enforced only when the target PAL device is active on channel c. Since the computations are performed over a finite field, we represent real-valued coefficients and thresholds using *decimal fixed-point* encoding: for a value $x \in \mathbb{R}$ we store the integer $\tilde{x} = \lfloor x \cdot 10^f \rceil$, where f is the chosen number of decimal digits of precision. Because circuit arithmetic is performed modulo a prime field, we apply range checks (bit-length bounds) to encoded inputs and to intermediate aggregates so that integer comparisons are well-defined and cannot be satisfied via modular wrap-around.

For Constraint 5, as shown in Eq. (5), we observe that it enforces a minimum distance between any two GAA CBSDs assigned to the same channel c within a county. For each unordered pair (n, g), the circuit computes a conditional lower bound $y^c_{n|j}\, y^c_{g|j} \left(R_{n|j} + R_{g|j}\right)$. Distances and transmission ranges are represented as bounded integers using fixed-point encoding, and then an inequality between $D_{a,b}$ and the lower bound is implemented.

For Constraint 6, as shown in Eq. 6, For each DPA point m, the circuit aggregates the weighted PAL and GAA, and gates the inequality with the binary indicator Z^c_m, so the constraint is enforced only when m is active on channel c. Similar decimal fixed-point encoding and range limit checking techniques as with Constraint 3 are used.

B. Additional Allocation Integrity Constraint

With the channel assignment indicators used as private inputs to the proof, the proof has a weakened soundness property, since an dishonest prover can send fake channel assignments other than the ones used to generate the proof, to the clients. In other words, a verifier cannot verify whether the allocation provided to clients is the original one. To resolve this issue, we impose an additional constraint on our circuit construction to enforce the integrity of allocation, leveraging cryptographic commitments as public inputs.

***Constraint** 7 (Allocation Integrity):* The channel assignments $\mathbf{alloc}$, as defined at the beginning of Section II-B and treated as a private input (witness), should satisfy a public commitment according to the following constraint:

$$\mathsf{com} = \mathsf{commit}(\mathbf{alloc}) \tag{7}$$

where com is a public input known to the verifier representing the precomputed commitment value; commit is the commitment function implemented within the arithmetic circuit.

The public commitment binds all allocation decisions to a single value, so that a prover cannot satisfy different constraints using inconsistent witnesses across separately generated proofs.

To concretely implement Eq. (7), we use a hash scheme $H : \mathbb{F}^L \to \mathbb{F}$ as the commitment function. $\mathbb{F}$ is a finite field and

$$L = A \cdot P + A \cdot G + M \tag{8}$$

where $A = |\mathcal{A}|$ is the number of counties; P and G are the numbers of PAL users and GAA users, respectively (we assume they are fixed, without loss of generality); M is the size of an auxiliary metadata vector meta that binds the commitment to the specific problem instance (i.e., specifying A, P, G, and the channel index c), preventing replay of the same allocation commitment across different instances.

First, we define a flattened vector $\mathbf{v}$ as

$$\mathbf{v} = \mathrm{Flatten}(\mathbf{alloc}, \mathrm{meta}) \in \mathbb{F}^L \tag{9}$$

where $\mathrm{Flatten}(\cdot)$ lists entries in the fixed order: $\{x\}$ in county-major and then user-major, followed by $\{y\}$ in county-major and then user-major, and followed by meta. Finally, the commitment is computed by

$$\mathsf{com} = H(\mathbf{v}) \tag{10}$$

Realization with efficient hash. We opt for Poseidon hash [23] due to its arithmetic-native design, which drastically reduces the circuit size (number of R1CS constraints) compared to traditional hash algorithms like SHA-256. We provide the Poseidon hash configuration in Appendix A.

C. Assembly of Proofs

We generate a bundle of Groth16 proofs [20] to certify that a single allocation witness $\mathbf{alloc} := \{\{x\}, \{y\}\}$ satisfies all constraint circuits, including the added Constraint 7. The proof generation consists of five steps:

- **Step 1: Circuit compilation.** For each constraint $k \in \{1, \ldots, 7\}$, we compile the corresponding Circom circuit into an R1CS representation and a WebAssembly (WASM) witness generator.
- **Step 2: Trusted setup.** We perform Groth16 setup in two phases: (i) a circuit-independent Powers-of-Tau ceremony, and (ii) a circuit-specific setup for each constraint circuit k producing a proving key $.zkey$, a verification key vk_k, and the corresponding verification logic $\mathbf{Verify}_k$.
- **Step 3: Witness generation.** For each proof instance, we construct private inputs and compute the witness w_k using the circuit's WASM witness generator. The public input includes com.
- **Step 4: Proof generation and bundling.** Using the witness w_k and proving key $.zkey$ for circuit k, the prover generates a Groth16 proof π_k and public signal $public_k = \mathsf{com}$. The final allocation validity proof π_{ACV} is written as:

$$\pi_{ACV} := \{(\pi_k, \text{public}_k, vk_k)\}_{k=1}^{7} \tag{11}$$

Verification. A verifier checks

$$\forall k: \quad \mathbf{Verify}_k(vk_k, \pi_k, public_k) = 1 \tag{12}$$

and that all $public_k$ share the same commitment value com. The verification is successful if all checks pass.

Parallelization. To speed up proof generation, we use a two-stage execution strategy in our implementation: (i) a serial phase that performs per-constraint compilation, trusted setup, and witness generation, and (ii) a parallel phase that runs Groth16 proving across constraints (one process per constraint), followed by off-chain parallel verification. Therefore, the wall time of the parallel proving phase is approximately determined by the slowest constraint prover thread, rather than the sum of all per-constraint proving times.

D. Complexity Analysis

Without the loss of generality, we assume each county has a fixed number P PAL users (=2 in CBRS), a fixed number N GAA users, $A = |\mathcal{A}|$ counties, the fixed number $F = |\mathcal{F}|$ channels (=15 in CBRS), and the length of the flattened commitment input vector is L, as defined in Eq. (8). Table I shows the computational complexity associated with each constraint. For Groth16, prover time is dominated by circuit size (number of R1CS constraints), which scales with the arithmetic and comparison/range-check operations in each circuit. As we will show in the experiments, Constraint 7 often dominates because its commitment has a large constant factor per permutation, so the end-to-end proving time is frequently close to the Constraint 7 proving time even though Constraint 7 scales only linearly in L.

In general, Groth16 verification is succinct: verifying one proof is independent of circuit size and costs a constant number of pairings plus a small linear term in the number of public inputs. Since our public inputs are small and per-proof verification is treated as constant, verification scales with the number of proofs.

TABLE I: Computational complexity associated with each constraint.

Item	Proof Gen	Proof File Size	Verification
Constraint 1	$\mathcal{O}(A)$	$\mathcal{O}(1)$	$\mathcal{O}(F)$
Constraint 2	$\mathcal{O}(A)$	$\mathcal{O}(1)$	$\mathcal{O}(A)$
Constraint 3	$\mathcal{O}(A^2 \cdot N)$	$\mathcal{O}(1)$	$\mathcal{O}(F)$
Constraint 4	$\mathcal{O}(A \cdot N)$	$\mathcal{O}(1)$	$\mathcal{O}(A)$
Constraint 5	$\mathcal{O}(A \cdot N^2)$	$\mathcal{O}(1)$	$\mathcal{O}(F)$
Constraint 6	$\mathcal{O}(A \cdot N)$	$\mathcal{O}(1)$	$\mathcal{O}(F)$
Constraint 7	$\mathcal{O}(L)$	$\mathcal{O}(1)$	$\mathcal{O}(F)$

IV. ZKPs for Move List Validity

In practical SAS deployment, upon incumbent arrivals, computing a new allocation faces tight time constraints. Instead, the move list mechanism can be used to decide which grants should suspended so that incumbents receive below-threshold aggregate interference.

A. ZKP Circuit Construction and Proof Assembly

Algorithm 1 shows the pseudocode of the standard move-list algorithm according to [24], augmented with our proof components. The basic idea is to prove the validity of each important step of the algorithm, rather than rewriting the entire algorithm into circuits. Some complex components, like sorting and interference calculation, can be done "off-circuit"; we directly use their outputs as ZKP witnesses.

Also, instead of proving the entire move list for all users (which is huge), we generate a proof personalized for each suspended user (i.e., those who are included in the move list).

The cutoff index $\hat{k}$, the threshold T, and the sorted interference list of all users of protection point p and channel c are treated as private witnesses. We first prove that the provided interference values are in nondecreasing order, so the user can trust that the list corresponds to a valid sorting outcome. Then the cumulative interference up to the cutoff is computed and the percentile condition: the aggregate value at $\hat{k}$ stays within the threshold, while adding the next element exceeds T, is proven, which establishes that $\hat{k}$ is the largest feasible cutoff. Finally, to enable personalized proofs, only the user's grant identifier is stated as a public witness, and we prove that the user is suspended exactly when their position in the sorted list is greater than $\hat{k}$. This lets the verifier confirm suspension correctness without learning any information.

We follow the same proof assembly pipeline as with the previous section on allocation constraints. For each suspended user, we compile the circuit, run the Groth16 trusted setup, generate a witness from the private inputs, and then produce and verify the corresponding proof under the exported verification key.

Algorithm 1 Standard move-list algorithm pseudocode [24] augmented with zkSAS's ZKP circuit generation steps (highlighted in blue fonts)

Input: Protection channel c, protection threshold t, full set of protection points $\mathcal{PP}$, full set of grants $\mathcal{GR}$.
Output: Move list for channel c denoted M_c ($\subseteq \mathcal{GR}$).

$M_c \leftarrow \emptyset$
for all point p in set $\mathcal{PP}$ **do**
  Find the set of grants in the "neighborhood" (within a certain radius) of protection point p and channel c: $G_{c,p} \leftarrow \text{Neighborhood}(\mathcal{GR}, c, p)$;
  Sort grants in $G_{c,p}$ by their median interference contribution to point p from smallest to largest: $(g_1, \ldots, g_n) \leftarrow \text{SortAscending}(G_{c,p}, \text{MedInt}(G_{c,p}))$;
  Proof strategy: use the sorted list as a private witness and prove its sortedness (not the sort algorithm itself).
  for all a from $minAzimuth$ to $maxAzimuth$ **do**
    Find the largest cutoff to the sorted list of grants such that the 95th percentile of the aggregate interference on a does not exceed the protection threshold t: $\hat{k} \leftarrow$ largest $k \in \{1, 2, ..., n\}$ s.t. $\text{AggInt95thprcntl}(\{g_1, ..., g_k\}, a) \leq t$;
    $\mathcal{M}_{c,p,a} \leftarrow \{g_{\hat{k}+1}, ..., g_n\}$;
    Proof strategy: generate a *personalized proof per user*. The cutoff index $\hat{k}$ is kept as a *private witness*, and we prove that it correctly satisfies the 95th-percentile threshold. Take the user's grant identifier as a *public witness* and prove that this identifier appears *after* the cutoff in the sorted list, which justifies the public claim for that user `is_suspended` $= 1$.
  end for
  $\mathcal{M}_{c,p} \leftarrow \bigcup_a \mathcal{M}_{c,p,a}$;
end for
$M_c \leftarrow \bigcup_p M_{c,p}$;

B. Complexity Analysis

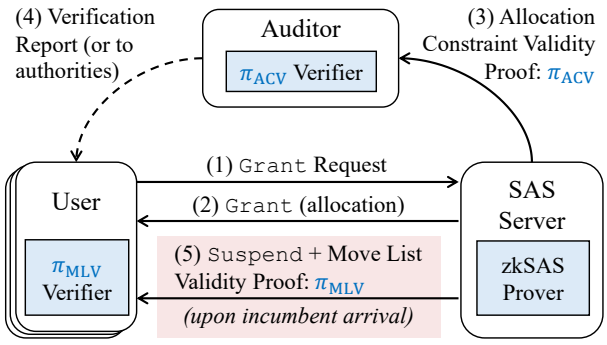
We assume that the size of the set of grants of a protection point p and channel c is G. Computation complexity for proof generation is $\mathcal{O}(G)$ since prover time is dominated by circuit size for Groth16. Also, as Groth16 verification does not scale with circuit size, computation complexity for proof verification is $\mathcal{O}(1)$.

V. zkSAS Use Cases

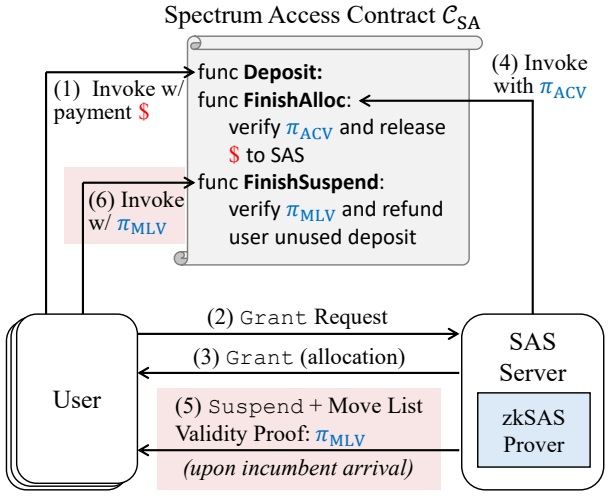
zkSAS is designed to serve as a modular verifiability tool for general SAS paradigms. In this section, we describe two primary use cases compatible with existing models, followed by a new use case for next-era spectrum sharing.

A. Auditing Centralized SAS Functions

zkSAS can be directly integrated into the current CBRS SAS implementation for checking the validity of grant generation and move-list inclusion. Fig. 1a shows an example workflow between a user, a SAS server, and an external auditor. After an initial inquiry (not shown in the figure), a user sends a `Grant` request to the SAS server (Step-1). The SAS server generates a new allocation, which comprises a list of grants for different users according to a certain algorithm, before sending the approved `Grant` back to each user (Step 2). The SAS server executes the zkSAS prover to generate the allocation constraint validity proof π_{ACV} according to the method described in Section III and sends it to the auditor for verification (Step 3), who routinely releases the verification report to users or a regulatory authority (Step 4). Notably, if there is no active incumbent during the allocation computation, π_{ACV} does not need to include proof of Constraint-6 for reduced overhead.



(a) Use case 1: auditing centralized SAS functions.



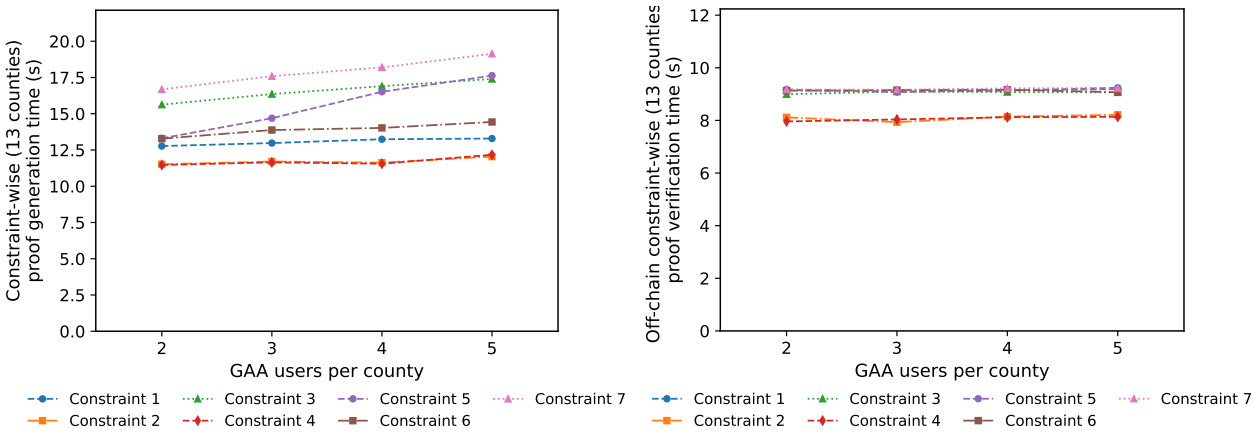
(b) Use case 2: supporting blockchain-based SAS.

Fig. 1: Two primary use cases of zkSAS.

When the SAS detects an incumbent arrival, it checks the up-to-date move list and sends the `Suspend` signal to users whose grants are in the move list (Step 5); this is accompanied by the move list validity proof π_{MLV} produced according to the method described in Section IV. Upon verifying π_{MLV}, the affected user vacates the band immediately.

B. Entrusting Off-chain Execution for Blockchain-based SAS

zkSAS can be integrated into a general blockchain-based SAS framework to address the privacy and efficiency gap,



(a) Constraint-wise proof generation

(b) Constraint-wise proof verification

Fig. 2: Constraint-wise timings of allocation constraint validity proofs (Total County = 13) vs GAA users per county.

as mentioned in Section I, when coupled with *off-chain execution*. Off-chain execution is an emerging computing paradigm in the blockchain/Web3 ecosystem that migrates an originally on-chain logic to an "off-chain" environment for private and efficient execution. Here, we use BD-SAS [6], a recent blockchain-based SAS framework, for example. The simplified workflow is shown in Fig. 1b. Instead of executing the grant request-allocation-payment logic entirely on the spectrum access contract $\mathcal{C}_{SA}$, we can offload the most computationally intensive step, allocation computation, to the off-chain server (an existing SAS server would suffice). A user initiates spectrum access by depositing the payment to $\mathcal{C}_{SA}$ (Step 1) and sending a `Grant` request to the off-chain server (Step 2). To claim the payment, the server needs to perform the allocation algorithm faithfully (Step 3) and upload the corresponding allocation constraint validity proof π_{ACV} to the contract, invoking the **FinishAlloc** function (Step 4). In an atomic step with this function, if π_{ACV} is verified, the payment will be released to the server. Invalid π_{ACV} will allow the user to claim back the deposit after a mandatory delay, which serves to mitigate request spamming from users (not shown in the figure).

Similar to the first use case, zkSAS's move list validity proof π_{MLV} can facilitate grant suspension upon detecting incumbent arrivals (Step 5). Vacated users are entitled to claim back unused deposits (due to shortened grant period) by uploading π_{MLV} to the contract invoking the **FinishSuspend** function (Step 6). If π_{MLV} is verified, the unused deposit will be sent back to the user.

C. Supporting "Spectrum Bux" in Next-gen Spectrum Sharing

To incentivize and monetize more efficient use of shared spectrum, next-generation spectrum sharing frameworks envision a market-based ecosystem where both incumbent and commercial users request short-term spectrum access by making Spectrum Bux (SBX) payments to a band manager [25]. In this model, SBX functions as a government-issued spectrum currency allocated to federal incumbent users, while Band manager, private-sector entitles analogous to SASs, provide allocation services. Aligned with this vision, band managers that provide Pay-As-You-Go (PAYG) services can leverages zkSAS to generate validity proofs for real-time channel assignments, similar to use case 2. Furthermore, for service models requiring anonymous channel acquisition, such as those involving military users, the zkSAS prover can be set up on the user side to generate zero-knowledge balance proofs. These proofs demonstrate sufficient SBX ownership, enabling band managers to authorize immediate service while deferring financial settlement. We will pursue these extensions in future work.

VI. Evaluation

A. Implementation

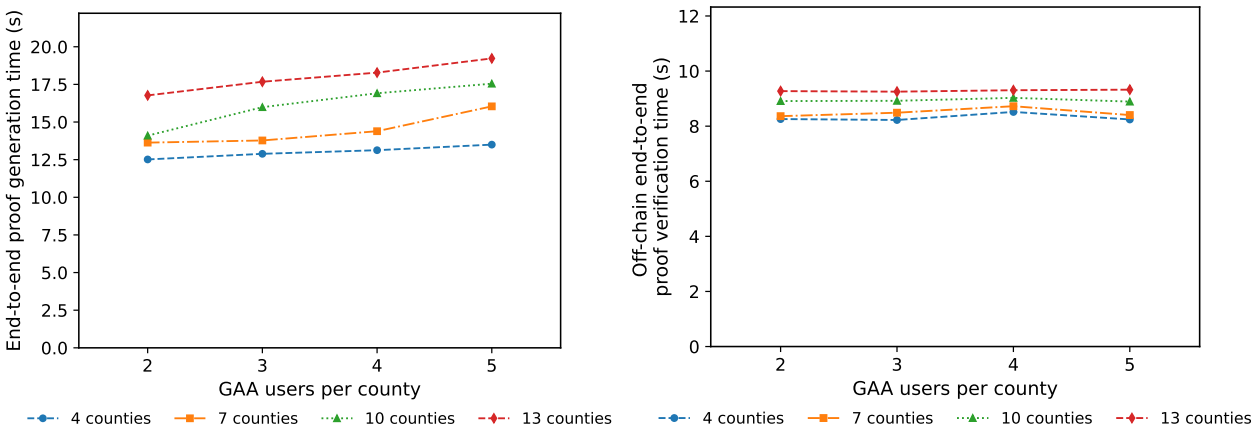
We implemented a prototype of zkSAS using `circom` (v2.1.x) and `snarkjs` (Groth16 [20]). Our implementation consists of the seven allocation-constraint circuits and the move-list validity circuit, totaling approximately 730 lines of circom code.[1] For Constraint 5 specifically, we represented non-integer inputs using fixed-point encoding with one digit after the decimal point.

We tested our implementation on two platforms. Proof generations were tested on a MacBook Pro laptop with Apple M4 Pro and 24 GB of memory. Proof verification was tested on the same MacBook Pro laptop and a locally deployed blockchain system based on Hyperledger Fabric [26]. Our Fabric environment ran on a single desktop server with Ubuntu 20.04 LTS and an Intel Xeon E5-2630 v3 @ 2.40 GHz CPU (32 logical processors).

B. Constraint-wise Computational Overhead

We first evaluate the generation and verification overhead of our allocation constraint valid proof (Section III) in our local laptop environment. The results are shown in Fig. 2. We

[1] Source code and artifact available at: https://github.com/purbasha-nishat/Spectrum-Allocation-Constraint-Circuits



(a) End-to-End proof generation

(b) End-to-End proof verification

Fig. 3: End-to-End timings of allocation constraint validity proofs vs GAA users per county.

observe that constraints 2 and 4 remain nearly constant as the number of GAA users per county increases. For the rest of the constraints, proof generation increases as the input size grows, but the growth rate varies widely by constraint. In particular, Constraint 5 exhibits the steepest increase, and it will become the dominant contributor to end-to-end proving time when the number of total counties and the number of GAA users per county increase.

Proof verification is comparatively stable across the same sweep, consistent with Groth16's verifier cost being largely insensitive to witness size. However, verification for constraints 2 and 4 scales with the total number of counties, since the number of public witnesses grows with the total number of counties.

C. End-to-end Computational Overhead

Next, we evaluate the computational overhead of the holistic proof generation and verification process for both the allocation constraint validity proof and the move list validity proof. Fig. 3 shows the results of allocation constraint validity proof. We observe that the end-to-end proof generation increases as the number of GAA users per county grows, and it also rises noticeably with the total number of counties (Fig. 3a). This is expected because increasing either dimension expands the overall R1CS circuit size, causing the slowest constraint in the bundle to dominate the total proving time. In comparison, end-to-end proof verification remains largely stable as the number of GAA users per county increases (Fig. 3b), consistent with Groth16 verification being mostly insensitive to witness size. However, the verification curves shift upward as the number of counties grows, making verification primarily a function of county count rather than per-county GAA user count.

The results of the move list validity proof are shown in Fig. 4. We observe that the proof generation time per suspended user increases as the grant-set size grows. In contrast, the verification time remains nearly constant across the same sweep, consistent with Groth16 verification being largely insensitive to the witness size.

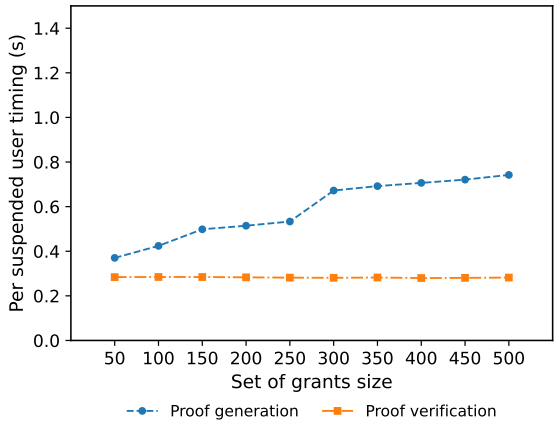


Fig. 4: End-to-end timings of move list validity proofs per suspended user timings vs set of grants size.

D. On-chain Verification Cost

We run the same verification steps on a Hyperledger Fabric (version 2.5.10) blockchain system with a dockerized network of 5 organizations, each with 1 peer node and 1 orderer node, deployed on a single server, a similar setup with BD-SAS [6].

Fig. 5 depicts the on-chain proof verification latency both for spectrum allocation constraints in different scenarios, and for the move list with a varying number of grants within a DPA. We measure the end-to-end latency for verifying the proofs of all constraints with varying numbers of counties and the number of users per county. The results show that the proof verification time is not affected by the SAS configuration or the complexity of the underlying constraints. The latency fluctuates between a range of 2.47 and 3.51 seconds, with no distinct upward trend correlated to the increase in the total number of GAA users. Notably, this on-chain verification takes shorter time than the off-chain counterpart (see Fig. 3b) mainly due to two reasons—the blockchain's server platform is more powerful than the laptop and the blockchain network size is

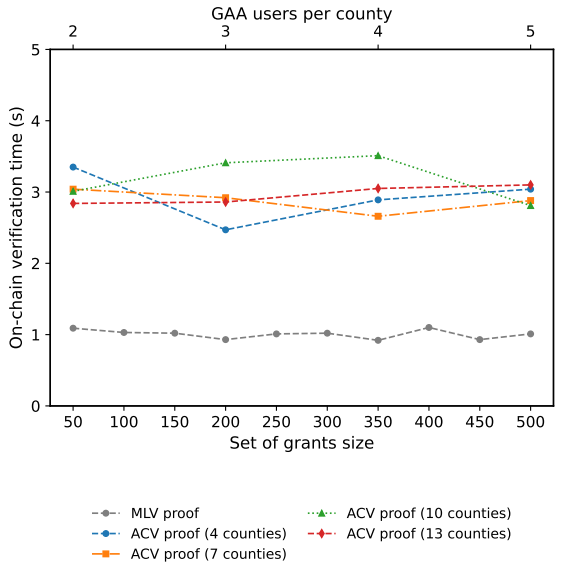


Fig. 5: On-chain verification timings. ACV: allocation constraint validity, varying with GAA users per county (top 4 curves). MLV: move list validity, varying with set of grant size (bottom curve).

relatively small (5 validators and 5 orderers).

Similar to the allocation proofs, the move list verification demonstrates a constant-time performance profile. Despite the significant increase in the number of grants, from 50 to 500, the proof verification runtime cost remains stable around 1 second, which is one block cycle and the shortest possible time for finalization. This clearly demonstrates that an additional ZKP verification layer introduces a fixed overhead, regardless of the scale of interference in a DPA. In both cases, the results also demonstrate the constant verification time, a property of Groth16, regardless of the witness size.

VII. Related Work

A. Alternative ZKP Systems

While our current implementation uses Groth16 [20] to maximize on-chain verification efficiency, the underlying zkSAS architecture is agnostic to the proof system and can be adapted to universal zk-SNARKs such as PlonK [27] and Marlin [19]. These protocols significantly reduce the burden of trusted setup by employing a single, updatable Structured Reference String (SRS) (instead of the per-circuit CRS in Groth16) usable across all circuits, with a marginal increase in prover overhead. Marlin [19] also supports R1CS constraints natively, facilitating direct integration with our existing design.

Furthermore, zkSAS may also be extended with transparent arguments that eliminate the trusted setup entirely, such as Bulletproofs [28], which relies on discrete logarithm assumptions; and STARKs [29], which utilize hash-based constructions to offer post-quantum security guarantees. However, adopting the transparent schemes for the current state of zkSAS necessitates a different arithmetization or algebraic representation of the constraints and performance trade-offs: Bulletproofs incur linear verification time, while STARKs generally produce significantly larger proof sizes compared to pairing-based alternatives. As a future research direction, transparent zk-SNARK schemes can be leveraged to address spectrum management systems for diverse SAS operators.

B. Blockchain-based Spectrum Access Management

Ariyarathna et al. [3] utilize smart contracts to automate the dynamic leasing of spectrum resources while ensuring Service Level Agreements (SLA). It also incorporates a spectrum token design to tokenize frequency rights. Zhang et al. [4] propose a blockchain-based SAS that employs a "Proof-of-Strategy" mechanism, which integrates the computation of optimal spectrum allocation directly into the block validation process. TrustSAS [5] and BD-SAS [6] are blockchain-based SAS designs that realize existing SAS functionalities, including channel inquiry, grant request handling, and grant issuance, with on-chain smart contracts. Both employ a two-level blockchain system design so that the global chain and the local chains handle different SAS tasks. TrustSAS further enables secondary users to query spectrum availability and commit to usage anonymously, while BD-SAS incorporates SAS server reshuffling to provide fault tolerance guarantees for each local chain. TriSAS [7] uses blockchain databases to realize a decentralized coordination process, as an alternative to the incumbent CPAS mechanism, to allow multiple independent SAS administrators to synchronize user inputs and agree on spectrum allocations without revealing their proprietary allocation algorithms. It further implements an on-chain verification module for checking the fairness and safety of proposed allocations.

VIII. Conclusion

We introduce zkSAS, a practical zero-knowledge proof system for providing verifiability of spectrum management operations. zkSAS can serve as standalone verification engine for existing SAS deployment to facilitate both external audits on allocation constraint satifiability and user audits on move list validity. It can be also integrated into blockchain-based SAS models coupled with off-chain execution to bridge the privacy and efficiency gap while establishing on-chain verifiability. Our evaluation shows that zkSAS achieves practical end-to-end latency at realistic county- and user-scale configurations, with verification remaining low-latency and proving time scaling predictably with instance size. We envision zkSAS as a valuable framework for establishing trust within next-generation spectrum sharing ecosystems.

Acknowledgment

This work was supported in part by the US National Science Foundation under grants 2433904, 2433905, 2331936, 232675, 2247560, 2247561; by the Office of Naval Research under grant N00014-24-1-2730; and by the Commonwealth Cyber Initiative.

REFERENCES

[1] The Office of the Federal Register (OFR) and the Government Publishing Office, "OFR: Electronic Code of Federal Regulations, Title 47: Telecommunication, Part 96 - Citizens Broadband Radio Service." https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.96, 2015.

[2] M. B. Weiss, K. Werbach, D. C. Sicker, and C. E. C. Bastidas, "On the application of blockchains to spectrum management," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 2, pp. 193–205, 2019.

[3] T. Ariyarathna, P. Harankahadeniya, S. Isthikar, N. Pathirana, H. D. Bandara, and A. Madanayake, "Dynamic spectrum access via smart contracts on blockchain," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, IEEE, 2019.

[4] H. Zhang, S. Leng, and H. Chai, "A blockchain enhanced dynamic spectrum sharing model based on proof-of-strategy," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[5] M. Grissa, A. A. Yavuz, and B. Hamdaoui, "Trustsas: A trustworthy spectrum access system for the 3.5 ghz cbrs band," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1495–1503, IEEE, 2019.

[6] Y. Xiao, S. Shi, W. Lou, C. Wang, X. Li, N. Zhang, Y. T. Hou, and J. H. Reed, "Bd-sas: Enabling dynamic spectrum sharing in low-trust environment," *IEEE transactions on cognitive communications and networking*, vol. 9, no. 4, pp. 842–856, 2023.

[7] S. Shi, Y. Xiao, C. Du, Y. Shi, C. Wang, R. Gazda, Y. T. Hou, E. Burger, L. DaSilva, and W. Lou, "Trisas: Toward dependable inter-sas coordination with auditability," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pp. 399–411, 2024.

[8] Federal Communications Commission (FCC), "Remarks of commissioner jessica rosenworcel [at] mobile world congress americas, los angeles, california, september 13, 2018." https://docs.fcc.gov/public/attachments/DOC-354091A1.pdf, accessed October 6, 2021, 2018.

[9] Wireless Innovation Forum, *Spectrum Sharing Committee Policy and Procedure Coordinated Periodic Activities Policy*, 6 2018. Version V1.3.0.

[10] K. S. Manosha, S. Joshi, T. Hänninen, M. Jokinen, P. Pirinen, H. Posti, K. Horneman, S. Yrjölä, and M. Latva-aho, "A channel allocation algorithm for citizens broadband radio service/spectrum access system," in *2017 European Conference on Networks and Communications (EuCNC)*, pp. 1–6, IEEE, 2017.

[11] S. Basnet, B. A. Jayawickrama, Y. He, and E. Dutkiewicz, "Fairness aware resource allocation for average capacity maximisation in general authorized access user," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, IEEE, 2018.

[12] X. Ying, M. M. Buddhikot, and S. Roy, "Sas-assisted coexistence-aware dynamic channel assignment in cbrs band," *IEEE Transactions on Wireless Communications*, vol. 17, no. 9, pp. 6307–6320, 2018.

[13] S. Basnet, Y. He, E. Dutkiewicz, and B. A. Jayawickrama, "Resource allocation in moving and fixed general authorized access users in spectrum access system," *IEEE Access*, vol. 7, pp. 107863–107873, 2019.

[14] N. Jai, S. Li, C. Li, Y. T. Hou, W. Lou, J. H. Reed, and S. Kompella, "Optimal channel allocation in the cbrs band with shipborne radar incumbents," in *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 80–88, IEEE, 2021.

[15] Wireless Innovation Forum, *CBRS Incumbent Protections and Encumbrances Overview*, 4 2020. Version V1.0.0.

[16] Wireless Innovation Forum, *Requirements for Commercial Operation in the U.S. 3550-3700 MHz Citizens Broadband Radio Service Band*, 10 2018. Version V1.6.0.

[17] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proceedings of the 3rd innovations in theoretical computer science conference*, pp. 326–349, 2012.

[18] M. Naor, "On cryptographic assumptions and challenges," in *Annual International Cryptology Conference*, pp. 96–109, Springer, 2003.

[19] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward, "Marlin: Preprocessing zksnarks with universal and updatable srs," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 738–768, Springer, 2020.

[20] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 305–326, Springer, 2016.

[21] snarkjs, "Javascript and pure web assembly implementation of zksnark." https://github.com/iden3/snarkjs.

[22] Circom, "Circom 2 documentation: Proving circuits." https://docs.circom.io/getting-started/proving-circuits.

[23] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for {Zero-Knowledge} proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 519–535, 2021.

[24] M. R. Souryal, T. T. Nguyen, and N. J. LaSorte, "3.5 ghz federal incumbent protection algorithms," in *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–5, IEEE, 2018.

[25] National Science Foundation, "NSF 24-549: Next Era of Wireless and Spectrum." https://new.nsf.gov/funding/opportunities/next-era-wireless-spectrum-newspectrum/nsf24-549/solicitation. [Accessed: 1/7/2026].

[26] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.

[27] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[28] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*, pp. 315–334, IEEE, 2018.

[29] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptology ePrint Archive*, 2018.

APPENDIX

A. Using Poseidon Hash

We use Poseidon hash [23] in our allocation integrity proof (Constraint 7) with the following parameter configuration: $ARITY = 16$ and $RATE = 15$. $ARITY$ is the number of input elements the hash function can accept in a single chunk; $RATE$ is the number of field elements that are "absorbed" in each permutation of the hash function. Let

$$R = \left\lceil \frac{L}{RATE} \right\rceil$$

be the number of absorption rounds. For each round $r \in \{0, \ldots, R-1\}$ and lane $j \in \{1, \ldots, RATE\}$, we define,

$$b_{r,j} = \begin{cases} v_{r \cdot RATE + j} & \text{if } r \cdot RATE + j \leq L, \\ 0 & \text{otherwise.} \end{cases}$$

where $b_{r,j}$ denotes the j-th element of the r-th absorbed message block. Initialize the chaining state as $s_0 = 0$ and update it for each absorption round r by

$$s_{r+1} = \text{Poseidon}\big(s_r, b_{r,1}, \ldots, b_{r,RATE}\big) \quad \forall r \in \{0, \ldots, R-1\}.$$

After R rounds, the final state s_R defines the Poseidon-based hash output,

$$H(\mathbf{v}) := s_R$$

and the circuit enforces commitment consistency by requiring

$$\mathsf{com} = s_R$$