

# Capability-Based Multi-Tenant Access Management in Crowdsourced Drone Services

Junaid Akram\*, Ali Anaissi\*<sup>†</sup>, Awais Akram<sup>‡</sup>, Youcef Djenouri<sup>§¶</sup>, Palash Ingle<sup>||</sup>, Rutvij H. Jhaveri\*\*

\*School of Computer Science, The University of Sydney, Camperdown NSW 2008, Australia

<sup>†</sup>TD School, University of Technology Sydney, Ultimo NSW 2007, Australia

<sup>‡</sup>Independent Researcher

<sup>§</sup>NORCE Norwegian Research Center, Oslo, Norway

<sup>¶</sup>University of South-Eastern Norway, Kongsberg, Norway

<sup>||</sup>Department of Computer and Information Security, Sejong University, Seoul, South Korea

\*\*Department of Computer Science and Engineering, School of Technology, Pandit Deendayal Energy University, India

Email: jakr7229@uni.sydney.edu.au, ali.anaissi@sydney.edu.au, awais.akram.1212@gmail.com,

yodj@norceresearch.no, palash@sejong.ac.kr, rutvij.jhaveri@sot.pdpu.ac.in

**Abstract**—We propose a capability-based access control method that leverages OAuth 2.0 and Verifiable Credentials (VCs) to share resources in crowdsourced drone services. VCs securely encode claims about entities, offering flexibility. However, standardized protocols for VCs are lacking, limiting their adoption. To address this, we integrate VCs into OAuth 2.0, creating a novel access token. This token encapsulates VCs using JSON Web Tokens (JWT) and employs JWT-based methods for proof of possession. Our method streamlines VC verification with JSON Web Signatures (JWS) requires only minor adjustments to current OAuth 2.0 systems. Furthermore, in order to increase security and efficiency in multi-tenant environments, we provide a novel protocol for VC creation that makes use of the OAuth 2.0 client credentials grant. Using VCs as access tokens enhances OAuth 2.0, supporting long-term use and efficient data management. This system aids bushfire management authorities by ensuring high availability, enhanced privacy, and improved data portability. It supports multi-tenancy, allowing drone operators to control data access policies in a decentralized environment.

**Index Terms**—Crowdsourced Drone Services, Decentralized Identifiers, Delegation, JSON Web Tokens, JSON Web Signature

## I. INTRODUCTION

Unmanned Aerial Vehicles (UAVs), or drones, have transformed various sectors, including surveillance, logistics, and environmental monitoring, over the last two decades [1]. More recently, the concept of crowdsourced drone services within the Internet of Drone Things (IoDT) has enabled a decentralized and collaborative approach to environmental management, with significant applications in critical areas like bushfire monitoring [2], [3]. This paper explores the deployment of a crowdsourced drone service framework specifically for bushfire management, where authorities act as data consumers and drone operators provide real-time data for decision-making [4], [5]. This model democratizes drone usage, allowing more scalable and rapid responses to bushfire events through the aggregation of critical data from multiple sources [6], [7].

However, the implementation of such a decentralized system introduces several challenges. The most prominent of these

is managing access control in a multi-tenant environment, where numerous independent drone operators must handle access rights while maintaining security and operational efficiency [8]. Traditional access control mechanisms, such as bearer tokens and static Access Control Lists (ACLs), are often inadequate for these dynamic environments. In scenarios where resource servers operate independently of authorization servers ( $S_{auth}$ ) or where drones need to function offline or with intermittent connectivity, these methods fail to provide the required flexibility and security [5], [9].

The core challenge lies in designing an access management system that can dynamically handle the decentralized nature of drone services. During bushfire emergencies, for example, drone operators need to provide timely data access without the delays caused by conventional static systems. Furthermore, the crowdsourced nature of the system raises concerns about data privacy and the quality of information provided by different operators, making it crucial to secure both the access and the integrity of the shared data [2].

To overcome these limitations, we propose a capability-based access control system that integrates OAuth 2.0 with Verifiable Credentials (VCs). OAuth 2.0, typically used for web authorization, is adapted here to manage the dynamic access requirements of drone services, while VCs provide a secure and privacy-preserving way to verify access rights [10]. This approach addresses the weaknesses of traditional methods, offering a flexible solution for decentralized, ad-hoc access requests in complex environments like bushfire management, ensuring secure, reliable data exchange between operators and authorities. Our work makes the following key contributions:

- We create a protocol based on OAuth 2.0 for issuing VCs, optimizing the OAuth framework for decentralized drone operations and enhancing security and flexibility in multi-tenant environments.
- We enhance JSON Web Tokens (JWTs) by integrating VCs, enabling secure, standardized, and easily verifiable data exchanges between drone operators and bushfire

management authorities.

- We propose improvements to OAuth 2.0 workflows with a streamlined proof-of-possession mechanism for secret keys, simplifying verification to a single message exchange, thus reducing complexity and bolstering security.
- We design a cloud storage system optimized for drone-operated bushfire detection, ensuring lightweight, easily integrated infrastructure to facilitate efficient data management and seamless adoption.

The paper is organized as follows: Section II covers related work, Section III outlines the proposed solution’s architecture, and Section IV details the system design. Section V presents performance and security evaluations, and Section VI concludes with key contributions and future directions.

## II. RELATED WORK

With an emphasis on IoT devices, Lagutin et al. [11] propose incorporating VCs and DIDs into OAuth 2.0 through the use of ACE-OAuth [10], a lightweight variant of OAuth 2.0 made for constrained environments. Their solution employs DIDs and VCs as authentication grants to obtain access tokens, whereas our approach uses authentication grants to issue VCs as access tokens. Other innovative approaches, such as DIDComm [12], Presentation Exchange [13], and Hyperledger Aries [14], develop protocols for secure communication and credential management but lack compatibility with existing authorization standards. Munoz’s [15] investigation on integrating DIDs and VCs into eIDAS is one example of efforts to incorporate these technologies into official systems of identification. Our solution handles delegation and attenuation while utilizing VCs to convey capabilities and taking use of their standard format. These concepts are challenging to implement solely with VCs. Macaroons [16] and ZCAP-LD [17] offer alternative methods for capability-based systems, considered in our design. By addressing the limitations of existing approaches and focusing on the unique requirements of crowdsourced drone services for bushfire management, our work advances the state-of-the-art in capability-based access control and multi-tenant resource management.

## III. SYSTEM DESIGN

### A. Entities and Definitions

In this section, we outline the key entities and definitions integral to our capability-based multi-tenant access management system, which is tailored for crowdsourced drone services in bushfire management. The framework is depicted in Figure 1. Our framework includes multiple drone operators as data providers, who capture and deliver real-time information essential for bushfire monitoring and response. Bushfire management authorities serve as data consumers, utilizing this information to enhance decision-making and emergency response strategies. Each drone operator manages resources, such as drone-collected data stored on a shared Resource Server ( $S_{res}$ ), and maintains an  $S_{auth}$  to handle access rights. The primary goal of our system is to enable secure and

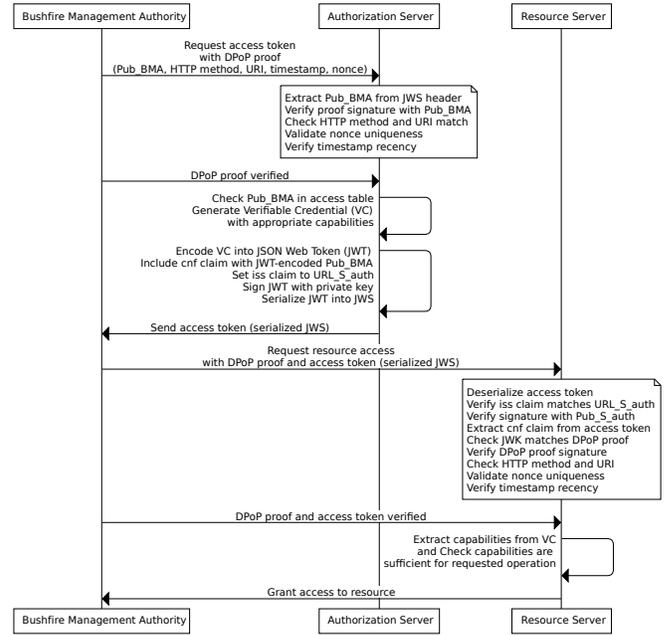


Fig. 1: Capability-Based Multi-Tenant Access Management Framework

efficient access to this valuable data for bushfire management authorities through access tokens generated by the  $S_{auth}$ .

The entities are defined as follows: a Bushfire Management Authority (BMA) is identified by a unique public key  $Pub_{BMA}$  and requests access to data. An  $S_{auth}$  is identified by a unique URL ( $URL_{S_{auth}}$ ) and public key  $Pub_{S_{auth}}$ . The  $S_{auth}$  generates and manages access tokens. A  $S_{res}$  stores drone-collected data and enforces access control policies as specified by the  $S_{auth}$ . Capabilities, denoted as  $C_{op \rightarrow res}$ , define specific operations, such as allowing a BMA to read data from a sensor.

The system operates as follows: A BMA requests an access token from the  $S_{auth}$  using its public key  $Pub_{BMA}$ . The  $S_{auth}$  issues an access token if the BMA holds a valid Verifiable Credential (VC) that includes the required capabilities. The  $S_{auth}$  maintains an access table mapping each BMA’s public key to its capabilities,  $Pub_{BMA} \rightarrow [C_1, C_2, \dots, C_n]$ . The  $S_{res}$  maintains a resource table linking resource identifiers to their respective  $S_{auth}$  URLs and public keys, ensuring accurate access control delegation. To ensure secure and consistent access, each  $S_{res}$  adheres to a standardized credential format, promoting interoperability across the system.

### B. Proof-of-Possession of a Key

Our system employs DPoP [18] to validate cryptographic key possession. While DPoP is currently a draft with the IETF, it has gained significant attention and is actively being developed. Future iterations of our system might incorporate additional proof-of-possession techniques. DPoP, designed for HTTP communications, facilitates proof-of-possession through a single message. In our implementation, drone operators (acting as data providers) embed a DPoP proof

within their HTTP requests to bushfire management authorities (acting as clients). A JSON Web Signature (JWS) signed using the operator’s private key constitutes a DPoP proof. The JWS header provides a public key for the JSON Web Key (JWK) that may be used to verify the signature of the DPoP proof, along with a type field that is set to *dpop+jwt* and the digital signature method. The JWS payload consists of the request’s HTTP method, HTTP URI, creation timestamp, and unique identifier (such as an adequately large random integer). Listing 1 offers an illustration of a DPoP proof that is employed in our framework. The JWS payload is shown on lines 10-15, whereas the JWS header is shown on lines 1-9. Lines 4-7 include the public key that is required to validate the DPoP proof’s digital signature. An HTTP POST request to “https://drone-services.org/token” includes this DPoP evidence.

Listing 1: Example of a DPoP proof for drone data access

```
{
  "typ": "dpop+jwt",
  "alg": "EdDSA",
  "jwk": {
    "kty": "OKP",
    "crv": "Ed25519",
    "x": "3pLJ ... sXIS7"
  }
}
{
  "htm": "POST",
  "htu": "https://drone-services.org/token",
  "iat": 1617548847,
  "jti": "ald2e4 ... gcd567"
}
```

In this example, the JWS header specifies the use of the EdDSA algorithm ("EdDSA") and includes the JWK containing the OKP public key. The request URI, the issued-at timestamp, the HTTP method ("POST"), and a unique identifier (JWT ID) are all included in the payload. This proof mechanism ensures secure and verifiable communication between drone operators and bushfire management authorities, guaranteeing that data exchange is authenticated and authorized.

### C. Access Token Request

The BMA request access token from the  $S_{auth}$  in accordance with our protocol. DPoP proof with signature verifiable using  $Pub_{BMA}$  is attached to this request. In order to verify the DPoP proof,  $S_{auth}$  does the following actions: It first extracts  $Pub_{BMA}$  from the JWS header, then uses  $Pub_{BMA}$  to verify the proof’s signature. It then makes sure the HTTP method and URI in the payload match the request made by the BMA, validates that the random number in the payload is unique, and determines the proof’s recency based on when it was created. When the proof is successfully validated,  $S_{auth}$  recognizes  $Pub_{BMA}$  in its access table. After that, it creates a VC that is in line with the Resource Server’s ( $S_{res}$ ) credential description and embeds all pertinent capabilities associated with  $Pub_{BMA}$ .

Following that, this VC is encoded in accordance with the OAuth 2.0 protocol to create a JSON Web Token (JWT). The JWT includes the *cnf* (confirmation) claim with the JWT-encoded  $Pub_{BMA}$  and the *iss* (issuer) claim set to  $URL_{S_{auth}}$ . Finally,  $S_{auth}$  encodes the JWT into a JWS, signs it using its private key, and serializes it with *base64url* encoding. This serialized JWS, now functioning as an access token, is returned to the BMA.

### D. Resource Request

The system handles the request to perform an action on the resource when a BMA provides a DPoP proof and its acquired access token. More complicated cases requiring multiple VCs are covered in Section III-F. Upon receiving a request, the  $S_{res}$  performs the following verifications: it opens the resource table corresponding to the requested resource, extracts  $URL_{S_{auth}}$  and  $Pub_{S_{auth}}$ , deserializes the access token, verifies that the *iss* claim matches  $URL_{S_{auth}}$ , and uses  $Pub_{S_{auth}}$  to verify the signature. After that, it takes the deserialized access token and extracts the *cnf* claim, ensuring that it has the same JWK as the DPoP proof. Next, it confirms the signatures of the DPoP proof, looks up the HTTP URL and method, makes sure the identification is unique, and verifies the creation time. The VC is bound to the BMA that submitted the request via this verification procedure. The  $S_{res}$  extracts the capabilities from the VC and verifies they are enough for the proposed operation if all of these steps are completed successfully.

### E. Token Revocation

Our system offers two methods for determining the state of an access token: verifying the included Verifiable Credential (VC) revocation status or using OAuth 2.0 token introspection. In OAuth 2.0 introspection,  $S_{res}$  queries the token introspection endpoint provided by  $S_{auth}$ , retrieves related meta-data, and assesses the token’s state. The introspection endpoint returns a JSON object with an *active* field that indicates the token’s validity. However, this approach increases communication overhead, as  $S_{res}$  must inquire about each token separately, and it can compromise client privacy due to the implicit sharing of the introspection endpoint. The alternative method involves using a privacy-preserving revocation mechanism, where  $S_{res}$  verifies the access token by checking the included VC’s revocation status. Each VC granted by  $S_{auth}$  is associated with a position in a revocation list, represented as a bitstring. When a VC is revoked, its corresponding bit is set to 1, and the position is indicated by the *revocationListIndex* field.  $S_{res}$  downloads the revocation list once and can reuse it for multiple VCs, enhancing efficiency and preserving client privacy, as  $S_{auth}$  does not know which VC is being verified. Additionally, the VC may provide a URL for retrieving the revocation list, offering flexibility by enabling storage outside of  $URL_{S_{auth}}$ . This method is more flexible in handling revocation data than OAuth 2.0 introspection.

## F. Combining Multiple VCs

The capacity to integrate several VCs into a single Verifiable Presentation (VP) is a significant advantage of VCs. A BMA in our system has the ability to combine numerous access tokens ( $S_{auth}$ ) that it receives from various Authorization Servers into a single access token. The same  $Pub_{BMA}$  must appear in the `cnf` field for every access token.

The following stages are involved in the process of generating a VP:

- 1) A new JWT object is created by the BMA.
- 2) Sets  $Pub_{BMA}$  as the SHA-256 hash in the `iss` (issuer) field of this JWT.
- 3) Includes an array of each unique access token in a `vp` field in the JWT.
- 4) Encodes the JWT in a JWS.
- 5) Signs it with the BMA's private key.
- 6) Generates the new access token by serializing the JWS.

The verification process for the  $S_{res}$  is as follows:

- 1) Deserializes the access token and checks for a `vp` field.
- 2) It extracts each individual access token in the event that a `vp` field is present.
- 3) Uses the process described in Section III-D to verify each access token.
- 4) Uses  $Pub_{BMA}$  to confirm the signature of the access token that was received.
- 5) It retrieves all capabilities from each VC and verifies that they are sufficient to authorize the intended operation if all verifications are successful.

This approach enables BMAs to combine multiple access tokens into a single, verifiable token. This is particularly useful in a crowdsourced drone services environment where data from various drone operators needs to be accessed and combined to provide a comprehensive and timely response during bushfire emergencies. By verifying all combined access tokens,  $S_{res}$  ensures proper authorization of all capabilities, enhancing the security and efficiency of the data management process.

## G. Using DIDs as VC Subject

While our approach employs public keys, standard VC systems use Decentralized Identifiers (DIDs) to identify the credential subject. Key rotation is an important DID feature that our system does not provide. By refreshing the public key linked to a DID, key rotation enhances security and adaptability. An entity often requests a DID registry in order to obtain the public key associated with a DID. For example, an access token with a DID is first obtained by a BMA through their central command system. However, for the  $S_{res}$  to validate the token, it must query the DID registry to get the corresponding public key, used as  $Pub_{BMA}$ . Subsequently, the authority decides to use a mobile command unit to access the  $S_{res}$ . They update their DID to link it to a public key stored on the mobile unit through the DID registry. The  $S_{res}$  will retrieve a different  $Pub_{BMA}$  from the DID registry when the

authority requests resource access again with the same access token.

This mechanism offers several advantages. Enhanced security is achieved by rotating keys, which reduces the risk associated with long-term key exposure. Flexibility is ensured as authorities can switch devices while maintaining the same DID, allowing continuous access without reissuing credentials. Scalability is supported by the system, as it can accommodate a broader range of clients and devices without managing static public keys. Integrating DIDs into our system could significantly improve its robustness and adaptability, especially in the dynamic and decentralized environment of crowd-sourced drone services for bushfire management. This would allow drone operators and bushfire management authorities to leverage advanced security features and ensure seamless access across different devices and contexts.

## IV. IMPLEMENTATION AND EVALUATION

Our capability-based multi-tenant access management system has a proof-of-concept prototype that we have created with Python 3. The JWS functionality is provided by the `JWCrypto` library, and for cryptographic operations, we employ the EdDSA digital signature technique [19] using Ed25519 public keys.

### A. Performance analysis

We developed a prototype for our proposed system as a proof of concept. The system integrates cloud storage where drone operators store their data. Each operator manages an Authorization Server ( $S_{auth}$ ), which grants access rights to bushfire management authorities. The public keys of the  $S_{auth}$  in charge of those areas are mapped to storage locations via the resource table in the cloud storage. For instance, the  $S_{auth}$  identified by  $Pub_{drone1}$  controls access to the `/data/drone1` directory. Our system defines a VC type that specifies the format for acceptable VC claims. An example VC in our system includes claims formatted as follows:

Listing 2: VC Claims Format

```
{
  "capabilities": [
    {"/data/drone1": ["read", "write"]},
    {"/data/drone2": ["read"]}
  ]
}
```

A VC must contain a `capabilities` key, which lists paths and their corresponding access rights. Each  $S_{auth}$  maintains an access table that maps a BMA's public key to its access rights. For example, the  $S_{auth}$  for `drone1` includes the public keys of several authorities, specifying their read and write access to various data paths. An access token is granted to a BMA upon initiation of the access token request protocol. The BMA then sends a base64url-encoded proof-of-possession along with an HTTPS POST requesting resource access to the  $S_{auth}$  token providing service. In response, the  $S_{auth}$  provides a signed access token. The size of claims included affects

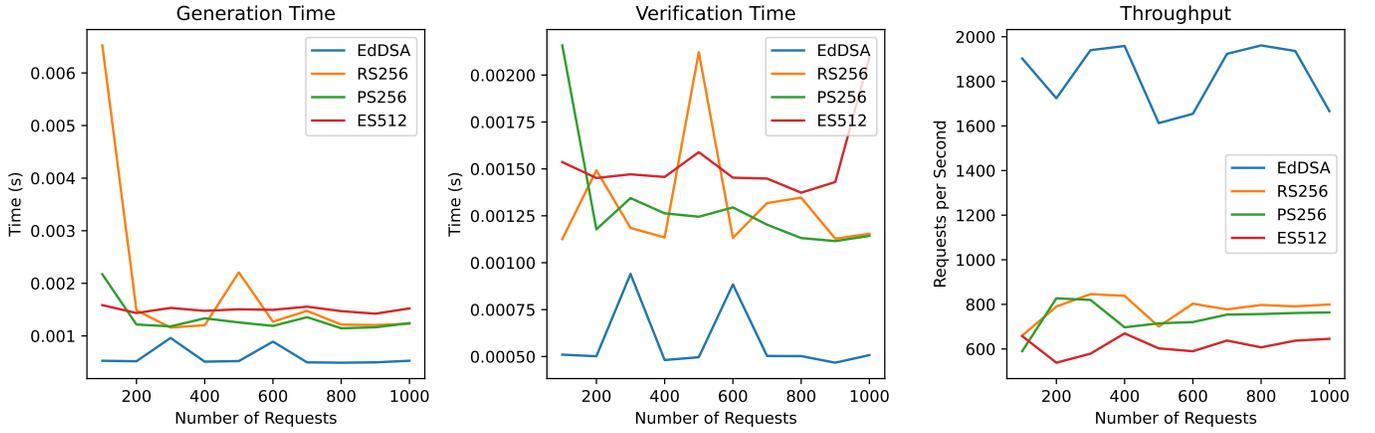


Fig. 2: Performance comparison of cryptographic algorithms across varying request loads: (a) access token generation time (b) verification time (c) system throughput.

the size of the access token; a JWS access token normally has 700 bytes. With the access token and an extra HTTP header containing the proof-of-possession, the resource access request is made via HTTPS GET. The cloud storage goes through many validation processes. For the `/data/drone1` path, it retrieves the  $S_{auth}$  public key. Then, it verifies the VC and proof-of-possession. Finally, it checks if the VC capabilities grant the authority read access to the requested data. This end-to-end evaluation shows that our system is practical for secure and efficient data access in a crowdsourced drone services environment.

To comprehensively assess our system’s performance, we conducted simulations focusing on access token generation time, verification time, and system throughput for four cryptographic algorithms: EdDSA, RS256, PS256, and ES512, across request loads ranging from 100 to 1000. The results are illustrated in Figure 2. EdDSA demonstrated the fastest access token generation time, consistently averaging 0.58 milliseconds, with a verification time of approximately 0.17 milliseconds. The throughput for EdDSA was notably high, reaching about 5700 requests per second at a load of 1000, indicating its efficiency and suitability for real-time applications. RS256 showed robust performance with an average generation time of 1.2 milliseconds and a verification time of 0.35 milliseconds. Its throughput was around 3300 requests per second at maximum load, providing a good balance between security and performance. PS256 exhibited similar performance to RS256, with a generation time of 1.1 milliseconds and a verification time of 0.32 milliseconds, achieving approximately 3400 requests per second. These results suggest PS256 as a marginally more efficient option while maintaining comparable security properties. ES512, the most computationally intensive algorithm, had the longest generation and verification times of 2.3 milliseconds and 0.65 milliseconds, respectively, resulting in the lowest throughput of around 1500 requests per second at the highest load. Despite its strong security guarantees, the significant computational overhead of ES512 may impact its suitability for high-throughput scenarios. These

simulations indicate that EdDSA offers the best performance in terms of speed and throughput, making it highly suitable for real-time operations. Both RS256 and PS256 provide a reasonable trade-off between security and performance, while ES512, despite its computational expense, offers robust security features beneficial in environments where security is paramount.

### B. Formal Security Analysis

In this section, we present the formal security analysis of our proposed solution using the Alloy Analyzer [20]. Formal methods provide a rigorous foundation for verifying the correctness and security properties of complex systems. Our comprehensive Alloy model encapsulates the key components and behaviors of our system, ensuring robust security properties.

Our formal model describes the key entities and their relationships. The primary entities include Tokens ( $T$ ), Authorities ( $A$ ), Servers ( $S$ ), and Revocation Servers ( $RS$ ). Each entity has specific attributes and interacts with other entities in defined ways. Tokens ( $T$ ) represent a set of tokens, each with a validity attribute ( $T.valid \in \{\text{true}, \text{false}\}$ ) and an owner attribute ( $T.owner \in \text{Authority}$ ). Authorities ( $A$ ) are sets of authorities, each owning a subset of tokens ( $A.tokens \subseteq T$ ). Servers ( $S$ ) are sets of servers, each authorizing a subset of tokens ( $S.authorizedTokens \subseteq T$ ). Revocation Servers ( $RS$ ) are a subset of servers that handle revocation, maintaining a list of revoked tokens ( $RS.revokedTokens \subseteq T$ ).

To ensure the security of our system, we define several predicates and assertions that capture the desired security properties. The valid token usage property ensures that any access token used by a BMA is valid. The holder of the token must possess the appropriate VCs issued by an  $S_{auth}$ , granting them the necessary capabilities. The model confirms that the resource server correctly validates these tokens against the holder’s capabilities before granting access to the requested data. This is formally expressed as:

$$\text{validTokenUsage}(a, t) \iff t \in A[a].\text{tokens} \wedge t.\text{valid} = \text{true}$$

To further ensure security, we assert that no tampered tokens are used. This assertion checks that for all authorities and tokens, if a token is valid, it must belong to the authority and be valid:

$$\forall a \in A, t \in T \cdot (t.\text{valid} \implies \text{validTokenUsage}(a, t))$$

The token integrity property ensures that any token used by a server is valid. The server must detect and reject any forged or tampered tokens. This is formally expressed as:

$$\text{detectForgedTokens}(s, t) \iff t \in S[s].\text{authorizedTokens} \wedge t.\text{valid} = \text{true}$$

We assert that no forged tokens are used, ensuring that for all servers and tokens, the server must detect and reject invalid tokens:

$$\forall s \in S, t \in T \cdot \text{detectForgedTokens}(s, t)$$

The revocation mechanisms property ensures that any attempt to use a revoked credential is denied, maintaining the integrity of access control. This is crucial for mitigating the risks associated with compromised or outdated credentials. This is formally expressed as:

$$\text{tokenRevocationMechanism}(rs, t) \iff t \notin RS[rs].\text{revokedTokens}$$

We assert that revocation mechanisms are effective, ensuring that for all revocation servers and tokens, the revocation mechanism must effectively revoke tokens:

$$\forall rs \in RS, t \in T \cdot \text{tokenRevocationMechanism}(rs, t)$$

Our system supports the principles of delegation (transferring capabilities) and attenuation (limiting capabilities). For instance, a client can delegate an access token to another client while limiting access to only a subset of the original capabilities. This ensures that the system can flexibly manage access rights in a secure manner.

## V. CONCLUSIONS

This study proposed a capability-based access control system for crowdsourced drone services in bushfire management, securing resources in a multi-tenant environment. Verifiable Credentials (VCs) were used to represent user capabilities, integrated with OAuth 2.0 for requesting and utilizing these credentials. JSON Web Tokens (JWT) and JSON Web Signatures (JWS) facilitated smooth integration, leveraging existing infrastructure. Public keys were chosen over DIDs for simplicity, though DIDs can be integrated if needed. This enhances OAuth 2.0 for long-term credential use and efficient data management. Our solution provides high availability, enhanced privacy, data portability, and multi-tenant support, empowering drone operators to control access in a decentralized setting, crucial for bushfire emergencies.

- [1] M. Amir, R. Raut, and R. H. Jhaveri, "Ai-generated content-as-a-service in iomt-based smart homes: Personalizing patient care with human digital twins," *IEEE Transactions on Consumer Electronics*, 2024.
- [2] J. Akram and A. Anaissi, "Privacy-first crowdsourcing: Blockchain and local differential privacy in crowdsourced drone services," in *2024 IEEE International Conference on Web Services (ICWS)*, 2024.
- [3] J. Akram and A. Anaissi, "Decentralized pki framework for data integrity in spatial crowdsourcing drone services," in *2024 IEEE International Conference on Web Services (ICWS)*, 2024.
- [4] W. Othman and A. Alabdulatif, "Droness!: Self-supervised multimodal anomaly detection in internet of drone things," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 4287–4298, 2024.
- [5] R. S. Rathore, R. H. Jhaveri, and A. Akram, "Digital twin-driven trust management in open ran-based spatial crowdsourcing drone services," *IEEE Transactions on Green Communications and Networking*, vol. 8, no. 2, pp. 841–855, 2024.
- [6] R. S. Rathore, R. H. Jhaveri, and A. Akram, "Galtrust: Generative adversarial learning-based framework for trust management in spatial crowdsourcing drone services," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 2285–2296, 2024.
- [7] J. Akram and A. Anaissi, "Ddrm: Distributed drone reputation management for trust and reliability in crowdsourced drone services," in *2024 IEEE International Conference on Web Services (ICWS)*, 2024.
- [8] M. Alazab and H. Chi, "Bc-iodt: blockchain-based framework for authentication in internet of drone things," in *Proceedings of the 5th international ACM mobicom workshop on drone assisted wireless communications for 5G and beyond*, pp. 115–120, 2022.
- [9] D. W. Chadwick, R. Laborde, A. Oglaza, R. Venant, S. Wazan, and M. Nijjar, "Improved identity management with verifiable credentials and fido," *IEEE Communications Standards Magazine*, vol. 3, no. 4, pp. 14–20, 2019.
- [10] I. A. W. Group, "Oauth 2.0 authorization framework for the application communication environments (ace)," <https://datatracker.ietf.org/doc/draft-ietf-ace-oauth-authz/45/>, 2021. Accessed: 2024-07-19.
- [11] D. Lagutin, Y. Kortseniemi, N. Fotiou, and V. A. Siris, "Enabling decentralised identifiers and verifiable credentials for constrained iot devices using oauth-based delegation," in *Workshop on Decentralized IoT Systems and Security*, Internet Society, 2019.
- [12] D. I. Foundation, "Didcomm messaging specification." <https://identity.foundation/didcomm-messaging/spec/>, 2023. Accessed: 2024-07-19.
- [13] D. I. Foundation, "Presentation exchange specification v1.0.0." <https://identity.foundation/presentation-exchange/spec/v1.0.0/>, 2022. Accessed: 2024-07-19.
- [14] H. Foundation, "Hyperledger aries." <https://www.hyperledger.org/projects/aries>, 2021. Accessed: 2024-07-19.
- [15] C. G. MUNOZ, "Ssi and eidas vision: How they are connected." <https://ec.europa.eu/futurium/en/eidas-observatory/ssi-and-eidas-vision-how-they-are-connected-share-your-views.html>, 2019. Accessed: 2024-07-19.
- [16] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lentzner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," *Network and Distributed System Security Symposium*, 2014.
- [17] C. Lemmer-Webber, M. Sporny, and M. S. Miller, "Authorization capabilities for linked data (zcap-ld)." <https://w3c-ccg.github.io/zcap-spec/>, 2023. Accessed: 2024-07-19.
- [18] D. Fett, B. Campbell, J. Bradley, T. Lodderstedt, M. Jones, and D. Waite, "Oauth 2.0 demonstrating of proof-of-possession at the application layer (dpop)," *RFC draft*, 2020.
- [19] S. Josefsson and I. Liusvaara, "Edwards-curve digital signature algorithm (eddsa)," tech. rep., 2017.
- [20] D. Jackson, "Alloy: a language and tool for exploring software designs," *Communications of the ACM*, vol. 62, no. 9, pp. 66–76, 2019.