Access Control: Advanced Authorization for the Digital Age

Anne Sai Venkata Naga Saketh

USC ID: 3725520208

Subject: CSCI530 Computer Security Systems

I have read the Guide to Avoiding Plagiarism published by the student affairs office. I understand what is expected of me with respect to properly citing sources, and how to avoid representing the work of others as my own. The material in this paper was written by me, except for such material that is quoted or indented and properly cited to indicated the sources of the material. I understand that using the words of others, and simply tagging the sentence, paragraph, or section with a tag to the copied source does not constitute proper citation and that if such materiel is used verbatim or paraphrased it must be specifically conveyed (such as through the use of quotation marks or indentation) together with the citation. I further understand that overuse of properly cited quotations to avoid conveying the information in my own words, while it will not subject me to disciplinary action, does convey to the instructor that I do not understand the material enough to explain it in my own words, and will likely result in a lesser grade on the paper.

Signed: 

University of Southern California

Revolutionizing Access Control: Advanced Authorization for the Digital Age

Sai Venkata Naga Saketh Anne
University of Southern California
annes@usc.edu

December 6, 2024

Abstract

Access control systems are vital for privacy, security, and efficiency in cloud environments. This paper focuses on state-of-the-art access control systems, particularly Amazon's Cedar and Google's Zanzibar architectures. Zanzibar is scalable and supports billions of users with granular access control, while Cedar introduces an ergonomic policy language for developers.

This study addresses limitations in traditional access control models and challenges in achieving low-latency policy checks. Explores trade-offs between security and performance in large-scale cloud environments.

The findings provide recommendations to improve the robustness of the access control system, including policy consistency, cross-cloud compatibility, and security breach detection and recovery mechanisms.

1 Introduction

Authorization and policy management are vital for ensuring privacy, security, and efficiency in digital ecosystems. Modern cloud-based applications, from online banking to photo storage, require robust mechanisms to manage resource access [10]. Traditional approaches embedding authorization logic into application code are error-prone, hard to audit, and lack scalability in distributed systems. As cloud adoption grows, dynamic and adaptable authorization solutions are essential.

Recent advancements like Amazon's Cedar and Google's Zanzibar address these challenges. Cedar externalizes access control logic using a policy-as-code language that enhances readability, safety, and performance [9]. Zanzibar, a global authorization system, manages permissions for billions of objects across Google services, offering scalability, low latency, and consistency [15]. These innovations enable unified architectures for seamless interoperability and improved user experiences.

Cloud platforms such as AWS, Azure, and GCP further enhance policy management with tools like IAM [3], Azure Policy [4], and Google Cloud Policy Manager [13]. However, achieving low-latency, high-availability, and globally consistent authorization remains challenging.

Frameworks like Zanzibar and Cedar align with zero trust architectures (ZTAs), which require dynamic, granular, and context-aware access control [19]. This research analyzes these systems' ability to scale, balance flexibility and performance, and secure cloud-native and zero trust environments, contributing to the development of efficient and adaptable authorization frameworks.

2 Background: Access Control Systems

Access control systems have advanced over decades, addressing scalability, expressiveness, performance, and policy validation. Foundational concepts from earlier systems, like Zanzibar and Cedar, have influenced contemporary systems.

Early Access Control Systems and Models: Access control emerged in early multi-user operating systems. Multics introduced Access Control Lists (ACLs) [20] for segments and directories, while UNIX [6] introduced file flags for owner and nonowner permissions. POSIX ACLs extended this to arbitrary lists of users and groups with up to 32 permission bits. VMS [16] introduced ACL inheritance for files within directory trees.

Role-Based Access Control (RBAC) and

Compound Identities: Several clients implement RBAC policies on a flexible namespace. The Taos system [8] extended ACLs with compound principals, enabling identity transformations in distributed environments [11].

Distributed Systems and Authorization in Social Graphs: As systems scaled, distributed authorization models became crucial. TAO, Facebook's distributed social graphs data store [8], which also manages social graphs, face similar challenges in scaling authorization checks. Both prioritize read-heavy operations and provide authorization in large-scale environments. While TAO offers eventual consistency. This design guarantees robust ACL and content update ordering, addressing issues like the new problem [1].

Commercial Identity and Access Management systems: IAM from companies such as Amazon [3], Google [13], and Microsoft [4] gained prominence in the late 2010s. These scalable cloud-based systems offered flexible access control mechanisms and unified ACL storage with RPC-based APIs.

Recent Advancements in Access Control: Research in access control has focused on distributed systems and sophisticated authorization models such as ReBAC. They load authorization models lazily for efficient evaluations and policy validation, similar to open-source alternatives like Ory Keto and AuthZed SpiceDB [2].

Symbolic Analysis and Performance of Authorization Languages: Recent research in symbolic analysis focuses on policy security and efficiency. Backes et al [5]. analyzed AWS IAM policies for mis-configurations, while Eiers et al [10], applied similar techniques to AWS IAM and Microsoft Azure policies.

Decentralized Authorization Approaches: Decentralized authorization systems like Cassandra explore trust management and multiple policy stores [12]. Cedar's future work may extend its design to support decentralized authorization, integrating its centralized, formal approach with emerging distributed models [7].

3 Comparision of Zanzibar and Cedar

Modern distributed systems need dynamic, scalable, and granular access control. Cedar's policy language is compared to XACML and OPA [17]. XACML supports ABAC and RBAC but has verbose syntax and limited performance. OPA is highly expressive. Zanzibar is compared to other access control systems like TAO and traditional ACL stores. Unlike TAO [1], which prioritizes eventual consistency, Zanzibar ensures external consistency and snapshot reads for reliable authorization checks at scale.

3.1 Key Differences Between Cedar and Zanzibar

Cedar and Zanzibar represent two distinct access control approaches, each tailored to specific use cases. This section compares them in terms of applications, advantages, disadvantages, and modes of operation.

3.1.1 Applications

- **Cedar:** Specialized access control system for fine-grained, policy-based access control. It's well-suited for dynamic and conditional access evaluations. Cedar uses a declarative JSON-based language, popular in cloud services like AWS, to define custom application policies [9].
- **Zanzibar:** Optimized for large-scale, graph-based access control, is widely used in social media, collaborative environments, and enterprise applications. It excels in managing hierarchical relationships, especially in distributed systems like Google, where it efficiently handles group memberships and content inheritance. [15]

3.1.2 Advantages

- **Cedar:**
  1. High expressiveness with conditional policies and dynamic constraints.
  2. Open-source and platform-agnostic, it's flexible for various environments.
  3. Low-latency policy evaluations optimized for cloud-native services.
- **Zanzibar:**
  1. External consistency through the zookie protocol ensures reliable operation ordering.

2. Scalability to trillions of access control tuples and millions of requests per second.
3. Unified data model simplifies integration across heterogeneous applications.

3.1.3 Disadvantages

- **Cedar:**
  1. Lacks built-in support for hierarchical relationships, requiring additional computation for group or resource inheritance.
  2. Limited ecosystem integrations compared to traditional access control systems.
- **Zanzibar:**
  1. System complexity due to advanced infrastructure like Google Spanner [14] and TrueTime.
  2. Challenges in open-source adoption due to the system design's close tie to Google's internal infrastructure.

3.1.4 Modes of Operation

- **Cedar:** Operates on a policy-based evaluation model with declarative access rules and dynamic evaluation. Policies support conditional grants, exclusions, and overrides for real-time access rule adjustments.
- **Zanzibar:** Uses a tuple-based graph model to define access relationships. Authorization checks traverse the graph using tuple rewrites to compute permissions, ensuring reliable evaluations with external consistency and snapshot isolation.

3.2 Implications for Zero Trust Architectures

The zero trust security model, based on "never trust, always verify," continuously validates user and device trustworthiness [19]. Zanzibar and Cedar are suitable for implementing zero trust principles.

1. **Dynamic Authorization:** Zanzibar's tuple-based evaluations enforce access rules in real-time, a key requirement in zero trust frameworks. Its distributed architecture ensures low-latency decision-making across globally dispersed resources [15].
2. **Context-Aware Policies:** Context-aware policies in Cedar's expressive language define dynamic rules sensitive to contextual parameters like geolocation, device posture, or time-of-day constraints, aligning with zero trust's adaptive security model [9].
3. **Scalability and Resilience:** Both Zanzibar and Cedar handle high authorization request volumes, crucial for enterprises adopting zero trust strategies to secure large-scale, diverse ecosystems.
4. **Enhanced Auditing and Compliance:** Enhanced auditing and compliance tools, provided by Zanzibar's changelog and Cedar's formal policy semantics, are crucial for implementing a zero trust model.

4 Architecture and Implementation

4.1 Zanzibar [15]

4.1.1 Overview: (Figure: 1)

Zanzibar's architecture is designed for high-performance, scalable access control across distributed systems. The core components are as follows:

1. **Access Control Servers:** The `aclserver` cluster manages operations like *Check*, *Expand*, and *Write*. These servers distribute requests across the cluster and perform boolean evaluations on Access Control List (ACL) tuples, maintaining strong consistency with tuple-based storage.
2. **Policy Storage Backend:** Access policies (ACLs) are stored in a global database system such as Google Spanner. The tuples are indexed by composite keys (e.g., object ID, relation, user) to support efficient hierarchical lookups. The backend ensures global availability and fault tolerance.
3. **Watch Servers:** These servers monitor changes in ACLs and propagate updates to clients in near real-time. They rely on

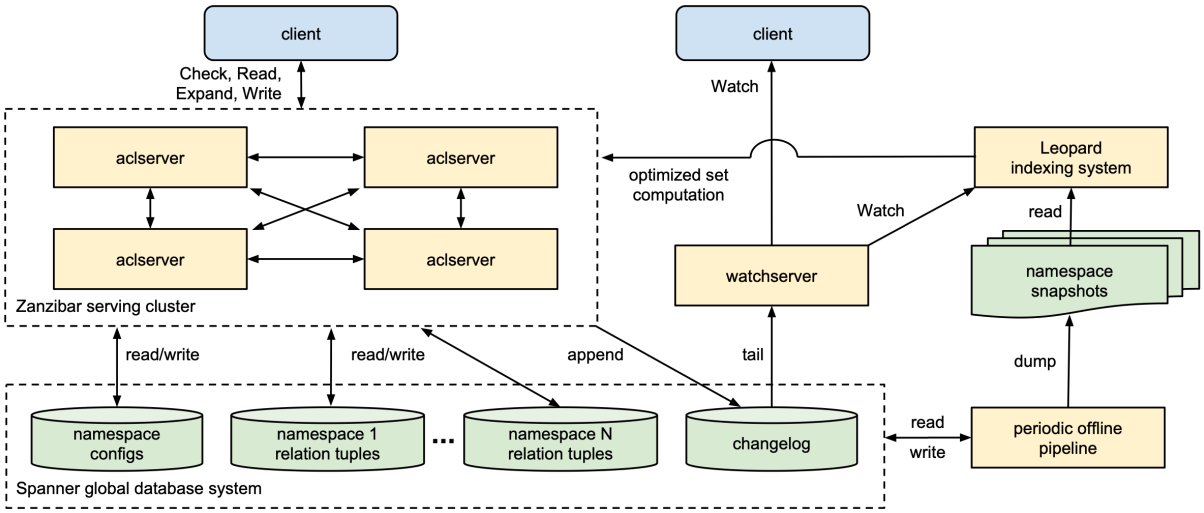

Figure 1: Zanzibar Architecture [15]

a dedicated `changelog` database for high-throughput updates, enabling consistent incremental change tracking.

4. **Indexing Layer:** Zanzibar employs a *Leopard Indexing System*, which builds offline namespace snapshots and handles incremental real-time updates. This supports efficient set operations for deeply nested group memberships.

5. **Caching Layer:** A distributed caching mechanism is integrated to reduce request latency by caching evaluation results and minimizing database interactions.

6. **Replication:** Zanzibar maintains regional replicas synchronized using Paxos-based consensus protocols. This ensures global consistency and low-latency access for distributed clients.

4.1.2 Key Features

1. **Consistency:** Zanzibar maintains strong consistency through evaluation timestamps, enabling dynamic snapshots of distributed access checks while prioritizing latency and ensuring correctness.

2. **Scalability:** The tuple-based model, augmented with sharding techniques, enables Zanzibar to achieve horizontal scalability, implementing hierarchical access control lists and intricate group memberships.

3. **Fault Tolerance:** Zanzibar employs mechanisms like request hedging, consistent hashing, and proactive monitoring to ensure reliability despite failures.

4.1.3 Implementation Details

1. **Check Requests:** Access checks are executed through recursive membership queries, optimized by the indexing layer.

2. **Expand Requests:** Expand requests traverse nested group memberships to generate comprehensive lists of users or groups with specific permissions.

3. **Write Requests:** These ensure ACL tuples are updated atomically across the storage backend and the change-log database, maintaining consistency during modifications.

4. **Watch Requests:** Watch requests stream incremental updates to clients in near real-time via the change-log database.

4.1.4 Optimizations

1. **Distributed Caching:** Distributed caching mitigates latency associated with recursive evaluations by storing intermediate results shared among requests.

2. **Nested Group Indexing:** Nested Group Indexing optimizes performance for deeply nested group structures using specialized data structures and optimized lookups.

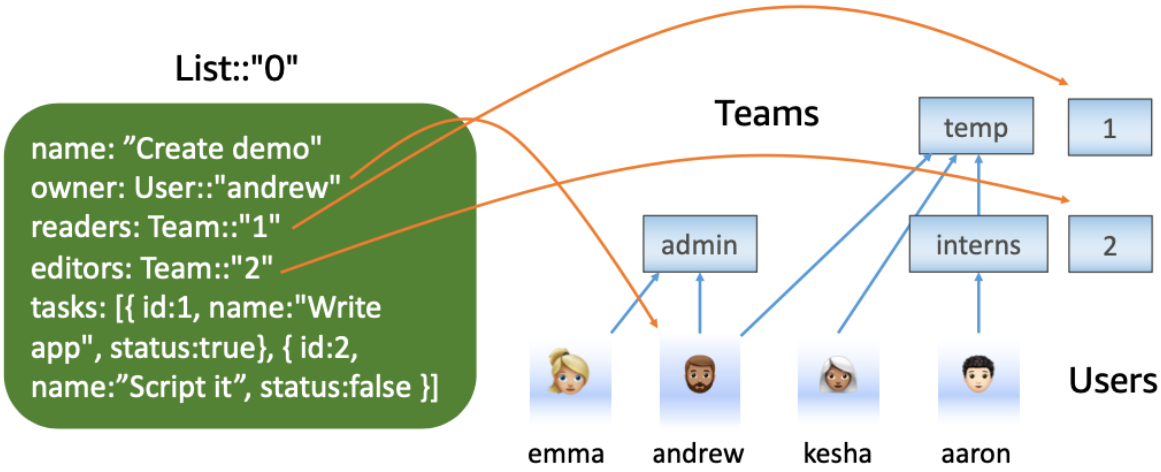

Figure 2: Cedar Architecture [9]

3. **Hot Spot Mitigation:** Dynamically identifies high-traffic objects and mitigates bottlenecks through caching and intelligent request distribution.

4.2 Cedar [9]

4.2.1 Overview: (Figure: 2)

Cedar's architecture uses a policy-based access control system that dynamically assesses permissions based on its declarative policy language. Key components include:

1. **Policy Model:** Cedar uses a structured policy model to define permissions for objects like Lists, teams, users, and their relationships. Policies are stored in a hierarchical format (e.g., JSON) and support dynamic evaluation.

2. **Access Control Evaluation:** Policies are evaluated in real-time using contextual data and relationships. For example, permissions for users (`emma`, `andrew`, etc.) are determined based on their roles in teams (e.g., `admin`, `interns`) and their association with resources (e.g., `tasks` in `List::"0"`).

3. **Policy Storage Backend:** Policies are stored in a hierarchical backend optimized for conditional checks and efficient queries. Tasks and permissions are linked to teams and users through structured relationships.

4. **Validation and Versioning:** Cedar incorporates tools to validate policy correctness and ensure version-aware consistency, preventing configuration errors and optimizing performance during concurrent updates.

4.2.2 Key Features

1. **Dynamic Policy Evaluation:** Cedar evaluates policies dynamically, leveraging contextual data like team memberships (`admin`, `interns`) and user relationships (`owner`, `readers`, `editors`).

2. **Expressive Policy Language:** Cedar's language allows complex boolean conditions and nested relationships, balancing expressiveness with safety for robust policy definitions.

3. **Cloud-Native Integration:** Cedar integrates seamlessly with cloud environments, utilizing built-in scaling, versioning, and performance enhancements.

4. **Policy Validation:** Built-in analyzers ensure policies are valid and adhere to safety and performance requirements, avoiding misconfigurations.

4.2.3 Implementation Details

1. **Policy Evaluation:** Policies are dynamically evaluated using structured expressions. Relationships like `readers`, `editors`, and `owner` link users and teams to resources for precise access control.

2. **Write Requests:** Modifications to policies (e.g., adding a user to `Team::"1"`) are applied atomically and consistently to maintain reliability.

3. **Validation Engine:** Cedar's validation tools prevent errors during updates by adhering to pre-defined safety and correctness criteria.

4.2.4 Optimizations

1. **Efficient Parsing and Evaluation:** Cedar optimizes policy evaluation by using indexing and hierarchical data structures to manage complex boolean conditions efficiently.

2. **Load Balancing:** Load balancing distributes requests across replicas for consistent performance under high load, especially in multi-tenant environments.

3. **Relationship Modeling:** Cedar excels at modeling complex relationships between users, teams, and resources, such as permissions for interns or *admins* across tasks in *"List::0"*.

5 Practical Usage and Real-Life Examples

5.1 Zanzibar

Real-Life Functionality: Zanzibar, a globally accessible global access control system designed by Google, enables applications to precisely define and enforce granular permissions for resources like documents, files, or services. Its tuple-based access control model with real-time evaluation allows dynamic and hierarchical access control within complex organizational structures.

Industry Adoption: Several companies have adopted Zanzibar's principles to construct their own access control systems.

1. **Google Workspace:** Zanzibar provides access control for Google Drive, Docs, and Calendar, allowing users to share resources with individuals or groups. For example, a shared document grants permissions to all team members.

2. **Authzed's SpiceDB:** Organizations use Authzed's SpiceDB, a commercial implementation of Zanzibar, to implement strict access control. For example, a ride-sharing company can restrict access to rider details to drivers assigned to specific trips.

3. **Airbnb:** Airbnb implements Zanzibar-inspired policies to enforce access control, host-guest communication, and property management.

Example: Zanzibar evaluates a user's request to modify a shared document within a content management system.

1. Whether the user is directly assigned "editor" permissions to the document.

2. Whether the user inherits permissions through group membership, like as a member of the "Editors" team.

The system retrieves and evaluates the ACL tuple for the document, ensuring real-time enforcement of access policies.

5.2 Cedar

Real-Life Functionality: Cedar, a policy-based access control system by AWS, manages permissions in applications and cloud environments. Its declarative language allows expressing dynamic conditions and contextual data, making it ideal for granular and flexible policies.

Industry Adoption: Cedar is now integrated into AWS services like Verified Permissions and is increasingly adopted for its expressive policy language.

1. **AWS Verified Permissions:** Cedar implements permissions management for customer-developed applications in AWS, including e-commerce platforms and HR systems, using dynamic user roles and resource access control.

2. **Custom Applications:** Developers use Cedar for dynamic access control, restricting sensitive data access based on user roles or predefined conditions like time of day.

Example: In a task management system, a user requests to modify a task. Cedar evaluates the request by:

1. Checking if the user belongs to a team assigned the "editor" role for the task.
2. Verifying contextual conditions, such as whether the task status allows modifications.

The policy evaluation determines access based on the conditions defined in Cedar's JSON policies.

6 Performance Evaluation Comparison

This section evaluates Zanzibar and Cedar, comparing their performance in managing complex access control scenarios.

6.1 Zanzibar Performance

Zanzibar was tested using synthetic workloads replicating production access patterns. Metrics included throughput, latency, and scalability in a sharded distributed system.

1. Achieved an average of 45,000 RPS at 15 ms latency with 500 clients, increasing to 50,000 RPS and 22 ms latency with 1000 clients. Latency scaled linearly with concurrency [15].
2. Caching resolved 80% of requests, significantly reducing response times. A cache hit ratio below 70% slightly increased latency, mitigated by optimized database queries [15].
3. Horizontal scaling added 8,000 RPS per shard without significant latency increases, validating scalability for large deployments [15].

6.2 Cedar Performance

Cedar was compared against OpenFGA and Rego [18] using real-world scenarios (Google Drive, GitHub, TinyTodo).

1. Cedar's median evaluation times were 4.0 μs (Google Drive), 11.0 μs (GitHub), and 5.0 μs (TinyTodo), outperforming OpenFGA (89–219 μs) and Rego (76–676 μs) [9].
2. Cedar scaled efficiently with minimal p99 increases, e.g., Google Drive's p99 time rose from 9.0 μs to 10.0 μs for 5 to 50 entities, while OpenFGA's grew from 283 μs to 3012 μs and Rego's from 391 μs to 1933 μs [9].
3. Policy slicing reduced Cedar's median evaluation time by 18% in template-based scenarios, though it incurred overhead compared to static policies [9].

7 Future Work

Future research in access control systems like Zanzibar and Cedar should focus on their integration into zero trust architectures (ZTAs) and leveraging AI/ML for enhanced security and scalability.

ZTAs require dynamic, context-aware access control. Enhancing Zanzibar's tuple-based evaluations and Cedar's policy language to support continuous validation of trust levels and cross-domain interoperability would make them essential for securing hybrid and multi-cloud environments.

AI/ML can automate policy creation, optimize scalability, and enhance anomaly detection. ML models could tailor policies based on usage patterns, while AI-driven systems could flag suspicious activities and forecast access patterns for dynamic resource allocation.

These innovations can align access control systems with the evolving demands of ZTAs, ensuring robust security, scalability, and adaptability.

8 Conclusion

Zanzibar and Cedar exemplify scalable, consistent, and flexible access control systems. Zanzibar's distributed architecture delivers low-latency, high-availability checks, managing trillions of rules and millions of requests per second with efficient group management and external consistency. While, Cedar provides an expressive, analyzable policy language with formal validation for sound policies, offering open-source flexibility and robust cloud integration for diverse use cases.

References

[1] M. Abadi et al. "A calculus for access control in distributed systems". In: *ACM Trans.*

Program. Lang. Syst. 15.4 (Sept. 1993), pp. 706–734.

[2] authzed-spicedb. *spicedb*. https://github.com/authzed/spicedb. Open Source, Google Zanzibar-inspired permissions database to enable fine-grained access control for customer applications. 2024.

[3] AWS Identity and Access Management (IAM). *Access Management – AWS Identity and Access Management (IAM)*. https://aws.amazon.com/iam/. 2024.

[4] Azure Policy Documentation. *Azure Policy Documentation*. https://learn.microsoft.com/en-us/azure/governance/policy/. 2024.

[5] John Backes et al. "Semantic-based Automated Reasoning for AWS Access Policies using SMT". In: *2018 Formal Methods in Computer Aided Design (FMCAD)*. 2018, pp. 1–9. DOI: 10.23919/FMCAD.2018.8602994.

[6] Bell Labs. *Unix Manual*. Accessed: 2019-04-16. 2019. URL: https://www.bell-labs.com/usr/dmr/www/pdfs/man22.pdf.

[7] Matt Blaze, Joan Feigenbaum, and Martin Strauss. "Compliance checking in the PolicyMaker trust management system". In: *Financial Cryptography*. 1998.

[8] N. Bronson et al. "TAO: Facebook's distributed data store for the social graph". In: *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC '13)*. 2013, pp. 49–60.

[9] Joseph W. Cutler et al. "Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization". In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1 (2024). To be published. DOI: 10.1145/XXXXXX.

[10] William Eiers et al. "Quacky: Quantitative Access Control Permissiveness Analyzer". In: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. Rochester, MI, USA: Association for Computing Machinery, 2023, pp. 1–5. DOI: 10.1145/3551349.3559530. URL: https://doi.org/10.1145/3551349.3559530.

[11] D. Ferraiolo and R. Kuhn. "Role-based access control". In: *15th NIST-NCSC National Computer Security Conference*. 1992, pp. 554–563.

[12] D. K. Gifford. "Information Storage in a Decentralized Computer System". AAI8124072. PhD thesis. Stanford, CA, USA: Stanford University, 1981.

[13] Google Cloud. *Cloud Identity and Access Management*. Accessed: 2019-04-16. 2019. URL: https://cloud.google.com/iam/.

[14] Google Cloud. *Cloud Spanner*. Accessed: 2019-04-16. 2019. URL: https://cloud.google.com/spanner/.

[15] Xueyuan Han et al. "Zanzibar: Google's Consistent, Global Authorization System". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2019, pp. 33–48. DOI: 10.1145/3341301.3359640.

[16] HP Development Company. *OpenVMS System Management Utilities Reference Manual*. Accessed: 2019-04-16. 2019. URL: https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c04622366.

[17] OASIS Standard. *Extensible Access Control Markup Language (XACML) Version 3.0*. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. Accessed: 2013. 2013.

[18] Open Policy Agent. *Rego Policy Language: Schema*. https://www.openpolicyagent.org/docs/latest/policy-language/#schema. Accessed: 2023. 2023.

[19] Scott Rose et al. *Zero Trust Architecture*. Special Publication NIST SP 800-207. Accessed: 2024-12-04. National Institute of Standards and Technology (NIST), 2020. URL: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf.

[20] The Open Group. *DCE 1.1: Authentication and Security Services*. Accessed: 2019-04-16. 2019. URL: http://pubs.opengroup.org/onlinepubs/9668899.