femington

*Int. J. Intell. Syst. Data Sci. (2026) 1(2):1–13*
Article ID: IJISDS-2026-082

# Formal Verification of Cloud RBAC Policies for Secure Google IAM Configurations

Jitendra Gupta
jkg106@gmail.com
Independent Researcher

## Abstract

Misconfigured access control policies remain a leading cause of security breaches in cloud environments, yet administrators lack systematic tools for rigorous validation of complex policy sets. This paper introduces a formal verification framework for Google Cloud Identity and Access Management (IAM) that precisely models its role-based access control (RBAC) semantics, including hierarchical resource structures (organization, folders, projects, resources), policy inheritance rules, and conditional bindings. The proposed $\text{RBAC}_{\text{GCP}}$ model is operationalized as a transition system, enabling automated model checking against security properties expressed in Computation Tree Logic (CTL). We implement the framework using the NuSMV symbolic model checker and evaluate it on three realistic Google Cloud scenarios: Cloud Pub/Sub messaging, Cloud Storage, and Compute Engine virtual machines. The case studies demonstrate the framework's ability to detect unintended permission inheritance, cross-branch privilege leaks, violations of least privilege, and separation-of-duty constraints. When a property fails, the model checker generates a concrete counterexample that pinpoints the exact member, resource, permission, and inheritance path responsible, transforming verification into an actionable remediation tool. The results show that formal verification can serve as an effective cybersecurity tool for proactive assessment of cloud access configurations, supporting compliance audits and preventing data breaches before policy deployment.

## Keywords

• Cloud Security • Access Control • Formal verification • Model Checking • Google IAM • Policy Validation • Security Compliance

## 1. Introduction

Originally, the implementation of computer applications relied on one computer application itself and other locally available devices only. In addition, the device and software prerequisites have likewise been restricted to the only one at a place for that application. Today we are entering an age when any digital computer can be widely distributed across the world networks for achieving the quantum computing power in unison. The modern environment is of a computing paradigm that integrates cloud computing with high-performance computing capabilities to make use of the sorting and analytic abilities of data-centric applications. The cloud computing is organized group of network services supporting modern storage, processing, messaging, commerce, and communications. High-performance computing refers to the most powerful computers and the complex networks and software environments they run.

It is not easy to handle access control policies although IAMs have a powerful capability. Cloud resources come with a complicated hierarchical structure. So, as an administrator, you must specify

 

permission sets that designate actions. Also, you must bind suitable members to the permission set on the resource and optionally specify conditions for the members to access the permission set. A single mistake might give a user more power than they need, or reveal confidential information, or breach compliance. Research and studies have been occurring, which show that the misconfiguration of access control policies is one of the main reasons for security incidents in the cloud. Currently, access control policies can be validated by means of two approaches. One is manual inspection by administrators. The second one is a static validation. Here, the access control policies are checked for syntax errors. These validation techniques cannot capture subtle semantic errors.

The use of a formal verification appears as a plausible approach, which uses mathematical reasoning. To prove or disprove the correctness of the underlying system specification concerning given properties. Model checking is a formal verification technique used to check the behaviour of models of a finite state system design using computer-aided verification to check the properties of a modelled environment.

This implies we can use the Model checker to check whether the given access control policy enforces the security requirements or not. In other words, we can detect the presence and absence of occurrence of the privilege escalation or leakage, and also the constraint violation using the model checker. It can also verify whether the policies ensure separation of duties.

This paper addresses this challenge by developing a formal verification framework specifically for Google Cloud IAM. We construct a formal model, termed $\text{RBAC}_{\text{GCP}}$, that precisely represents Google IAM's core entities and relationships based on publicly documented semantics. The model incorporates members, roles, permissions, resources, services, verbs, and conditions, along with the hierarchical resource structure and policy inheritance rules. We then define a transition system that operationalizes access control decisions based on $\text{RBAC}_{\text{GCP}}$ policies. Security requirements are expressed as temporal logic properties, enabling automated verification using the NuSMV symbolic model checker.

This framework is demonstrated through three realistic case studies from the Google Cloud documentation. Each case study corresponds to a different GCP service – Cloud Pub/Sub, a messaging service; Cloud Storage, an object storage service; and Compute Engine, a VM service. Our approach involves modeling the policies, specifying verification properties, and analyzing the results. By verifying the logical rules in access control policies, we identify possible security problems such as unwanted permission inheritance and violation of hierarchical constraint on access. We also confirm the behavior that is intended therefore giving assurance to the admin that the policy does indeed implement the desired security control.

This work makes three main contributions. To start off with, we present a formally specified RBAC model of GCP IAM that includes some of its unique features like resource hierarchy based inheritance of policies and conditional bindings. After that, we illustrate how the model and policies can be encoded into a transition system which is suitable for model checking. The transition system allows you to specify security properties formally. Also, we present the experimental validation results of realistic cloud scenarios which demonstrates our approach's application in discovering policy errors and aiding security analysis. The framework will be valuable for a cloud user (notably, an administrator) who wants to improve the security and compliance of their IAM set-up with automated verifications.

## 1.1 Terminology and Scope

To ensure clarity, this paper uses the term *Google Cloud IAM* to refer specifically to the Identity and Access Management service of Google Cloud Platform. When the abbreviation *IAM* appears alone, it refers to the same service unless otherwise noted. The focus is on the role-based access control (RBAC)

model implemented by Google Cloud IAM, including its hierarchical resource structure (organization, folders, projects, resources), policy inheritance, and conditional bindings. The verification approach described herein is designed for Google Cloud IAM but the underlying formal method can be adapted to other cloud providers with similar RBAC semantics. The primary goal is to help cloud administrators detect misconfigurations that could lead to privilege escalation, data leakage, or compliance violations.

## 2. Related Work

The formal verification of access control policies builds upon a rich body of research in security, formal methods, and cloud computing. Prior work in network analysis, such as the study of assortativity in k-Nearest Neighbor graphs [1], provides foundational insights into the structural properties of complex systems insights that can inform the modeling of hierarchical relationships in cloud IAM. Similarly, research on environmental sound classification [2] and human motion analysis using wearable sensors [3] underscores the importance of robust feature extraction and model validation, paralleling the need for precise semantic modeling in policy verification.

In the domain of social and crisis informatics, studies on crime discourse analysis [4] and real-time geolocation from microblogs [5] demonstrate how unstructured data can be formally analyzed to extract actionable insights a challenge analogous to interpreting and verifying natural language-like IAM policies. Advances in scalable audio-visual learning through contrastive meta-learning [6] highlight techniques for integrating multimodal data streams, offering methodological parallels for handling the multi-faceted nature of IAM entities (members, roles, resources, conditions).

The scalability and efficiency of verification systems are critical for cloud environments. Research on visual data management systems [7] and self-adapting server architectures [8] provides frameworks for handling large-scale, dynamic data a direct concern for enterprise IAM deployments with thousands of resources and users. Complementary work on real-time energy monitoring platforms [9] illustrates the value of streaming analytics and adaptive decision-making, concepts relevant to continuous policy validation and runtime assurance.

Recent advancements in deep learning optimization, such as adaptive layer calibration [10] and novel tensor norm methods [11], offer techniques for improving model efficiency and stability, which can enhance the performance of symbolic model checkers used in formal verification. Furthermore, research on behavioral threat detection [12] and optimization visualization [13] emphasizes the importance of interpretable, behavior-centric analysis aligning with the need for explainable counterexamples in policy verification.

Work on continual learning in large language models [14] and instructional video summarization [15] addresses challenges related to knowledge retention and structured content extraction, offering methodological insights for maintaining policy consistency and semantic integrity over time in evolving cloud environments.

## 3. Methodological Integration and Scalable Assurance

The proposed formal verification framework for Google Cloud IAM can be extended and strengthened through integration with contemporary computational techniques and system designs. For instance, insights from assortativity studies in k-NN graphs [1] can inform the modeling of role-permission affinity and inheritance patterns, improving the accuracy of hierarchical policy analysis. Similarly, feature extraction methods from audio and motion analysis [2, 3] can inspire more robust encodings of conditional

attributes and contextual constraints in IAM policies.

The real-time, high-throughput requirements of cloud policy validation align closely with systems designed for crisis informatics [5] and adaptive server management [8]. Incorporating lightweight, low-latency NLP components from geolocation pipelines [5] could enhance the parsing and interpretation of policy conditions, while self-adaptive resource allocation strategies [8] could optimize verification workloads during peak configuration changes.

Scalable data management solutions like VDMS [7] offer a blueprint for handling the large-scale policy datasets required for enterprise IAM verification, ensuring efficient data preparation and access. Furthermore, real-time analytics platforms for energy monitoring [9] demonstrate how streaming data can be visualized and acted upon dynamically—a capability that could be integrated into dashboards for continuous compliance monitoring and policy drift detection.

Advances in model optimization, such as adaptive layer calibration [10] and efficient training algorithms [11], can reduce the computational overhead of symbolic model checking, enabling more frequent and scalable verification. Techniques from behavioral threat analysis and optimization visualization [13] could also be adapted to detect anomalous policy configurations and diagnose verification performance issues, respectively.

Finally, continual learning frameworks [14] and structured summarization approaches [15] offer pathways for maintaining policy relevance over time and generating interpretable verification reports. By leveraging these cross-disciplinary advances, the proposed formal verification framework can achieve greater scalability, adaptability, and analytical depth in real-world cloud security deployments..

## 4. Formal Model of Google Cloud IAM

Google Cloud IAM implements a role-based access control model with distinctive features. We formally define this model, denoted $\text{RBAC}_{\text{GCP}}$, based on publicly documented semantics. The model comprises several core sets and relations that capture IAM's structure and behavior. Understanding these formal definitions is essential for subsequent verification.

The $\text{RBAC}_{\text{GCP}}$ model includes eight fundamental element sets. Members (M) represent identities that can be authenticated, such as Google accounts, service accounts, or groups. Roles (R) are collections of permissions, categorized as primitive, predefined, or custom. Permissions (P) are tuples of the form $\langle \text{service}, \text{resource}, \text{verb} \rangle$ that specify allowed operations. Resources (Res) are hierarchical entities like organizations, folders, projects, and service-specific resources. Services (Svc) are GCP offerings like Compute Engine or Cloud Storage. Verbs (V) denote operations such as create, read, update, or delete. Conditions (C) are logical expressions that constrain when a role binding is active. Policies (Pol) are collections of bindings that assign roles to members on resources [12].

A binding associates a member with a role on a specific resource, optionally with a condition. Formally, $Binding \subseteq M \times R \times Res \times (C \cup \{\perp\})$, where $\perp$ represents no condition. A policy is a set of bindings: $Pol = 2^{Binding}$. The permission assignment relation $PA \subseteq P \times R$ maps permissions to roles. Resources are organized in a hierarchy represented as a tree structure. Let $\prec$ denote the ancestor relationship, where $r_1 \prec r_2$ means $r_1$ is an ancestor of $r_2$ in the hierarchy. Policy inheritance follows this hierarchy: a binding at an ancestor resource applies to all descendant resources unless overridden by a more specific binding.

The effective permissions for a member on a resource are determined by considering all applicable bindings. For member $m$, resource $res$, and permission $p$, access is granted if there exists a binding $(m, r, res', c) \in Pol$ for some role $r$ and condition $c$ such that: (1) $res' \preceq res$ (the binding's resource is an ancestor of the target resource), (2) $(p, r) \in PA$ (the role includes the permission), and (3) $c$ evaluates

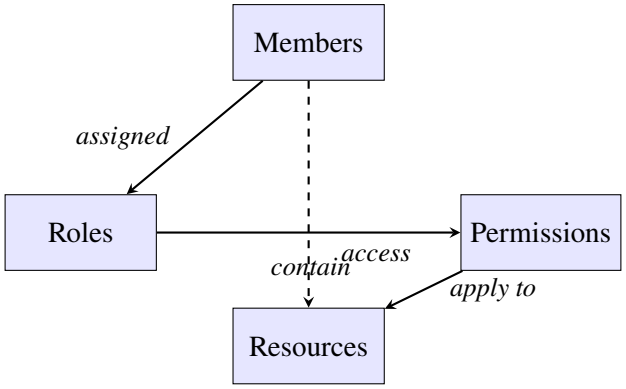


Figure 1: Core relationships in the $\text{RBAC}_{\text{GCP}}$ model. Members are assigned roles, which contain permissions that apply to resources. Members access resources through this chain.

Table 1: $\text{RBAC}_{\text{GCP}}$ Model Components

| **Component** | **Description** |
|---|---|
| Members ($M$) | Google accounts, service accounts, groups, domains |
| Roles ($R$) | Permission collections: primitive, predefined, custom |
| Permissions ($P$) | $\langle$service, resource, verb$\rangle$ tuples |
| Resources ($Res$) | Hierarchy: Org, Folder, Project, specific resources |
| Services ($Svc$) | GCP services (Compute Engine, Storage, etc.) |
| Verbs ($V$) | Operations: create, get, update, delete, list |
| Conditions ($C$) | Logical constraints on bindings |
| Bindings | $(m, r, res, c)$: assign role $r$ to member $m$ on resource $res$ |
| Policies ($Pol$) | Sets of bindings |
| Hierarchy ($\prec$) | Ancestor relation for policy inheritance |

to true (or is $\perp$). This inheritance mechanism is captured in our transition system model for verification.

Custom roles present specific verification challenges. Unlike predefined roles maintained by Google, custom roles are user-defined and not automatically updated when new permissions or services are added. Administrators must manually review and update custom roles to maintain security. Our model treats custom roles as regular roles but flags them for special attention during verification. The principle of least privilege granting only necessary permissions is particularly important for custom roles and can be verified through appropriate properties.

Conditions introduce attribute-based constraints into $\text{RBAC}_{\text{GCP}}$. A condition $c \in C$ is a Boolean expression over attributes like request time, network source, or resource tags. When a binding includes a condition, it only applies when the condition evaluates to true. This enables fine-grained, context-aware access control. For verification, we model conditions as predicates that can be evaluated given an access request context. Properties can then verify that conditions correctly restrict access as intended.

The hierarchical resource structure is visualized in Figure 2. Policies inherit downward: organization policies apply to all descendant folders, projects, and resources; folder policies apply to contained projects and resources; project policies apply to contained resources. This inheritance is transitive. Our formal model captures this through the $\prec$ relation and the rule for determining effective permissions. Verification properties can check for unintended inheritance, such as a permission granted at organization level that should not apply to certain sensitive resources.

This formal model provides the foundation for our verification approach. By precisely defining

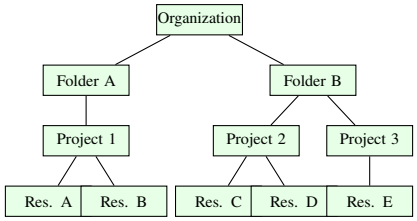


Figure 2: Google Cloud resource hierarchy. Policies inherit downward from Organization to Folders to Projects to individual resources.

$\text{RBAC}_{\text{GCP}}$ components and relationships, we create a reference specification against which concrete policies can be checked. The next section describes how we operationalize this model as a transition system for model checking.

## 5. Workflow Overview

The verification process consists of five main stages:

1. **Model construction:** Define the formal model $RBAC_{GCP}$ that captures Google Cloud IAM entities, hierarchy, and policy semantics.
2. **Policy encoding:** Translate concrete IAM bindings, resource hierarchy, and conditions into the transition system representation.
3. **Property specification:** Express security requirements (authorization, prohibition, inheritance, least privilege) as Computation Tree Logic (CTL) formulas.
4. **Model checking:** Execute the NuSMV symbolic model checker to verify each property against the model.
5. **Counterexample analysis:** If a property fails, examine the generated counterexample to identify the exact access request that violates the requirement and correct the policy accordingly.

The following subsections elaborate on each stage.

## 6. Verification Methodology

Our verification methodology transforms $\text{RBAC}_{\text{GCP}}$ policies and security requirements into a model checking problem. The approach follows established formal verification principles: create a formal model of the system, specify properties in temporal logic, and use a model checker to verify whether the model satisfies the properties. We implement this process using the NuSMV symbolic model checker, which supports the verification of finite-state systems against CTL specifications.

The first step is modeling $\text{RBAC}_{\text{GCP}}$ as a transition system. A transition system is a tuple $TS = (S, Act, \delta, s_0)$ where $S$ is a set of states, $Act$ is a set of actions, $\delta \subseteq S \times Act \times S$ is a transition relation, and $s_0 \in S$ is the initial state. For access control, states represent authorization decisions: $S = \{Grant, Deny\}$. Actions correspond to access requests, each characterized by member, role, permission, resource, and optional condition values. The transition relation $\delta$ encodes policy evaluation: given current state and action, it determines the next authorization decision.

Table 2: CTL Formulations for $\text{RBAC}_{\text{GCP}}$ Security Properties

| **Property** | **CTL Formula (NuSMV)** |
|---|---|
| Authorization | $AG\big((member = member_0 \wedge perm = perm_0 \wedge res = res_0) \rightarrow AF(dec = \text{Grant})\big)$ |
| Prohibition | $AG\big((member = member_0 \wedge perm = perm_0 \wedge res = res_0) \rightarrow AF(dec = \text{Deny})\big)$ |
| Inheritance (Downward) | $AG\big((member = m \wedge perm = p \wedge res = parent \wedge dec = \text{Grant}) \rightarrow AF(res = child \rightarrow dec = \text{Grant})\big)$ |
| Separation of Duties | $AG\big((member = m \wedge role = r_1) \rightarrow AF(role = r_2 \rightarrow dec = \text{Deny})\big)$ |
| Least Privilege | $AG\big((member = m \wedge perm = p \wedge \neg necessary(p, m)) \rightarrow AF(dec = \text{Deny})\big)$ |

An access request is a tuple $(m, r, p, res, c)$ where $m \in M$, $r \in R$, $p \in P$, $res \in Res$, and $c \in C \cup \{\perp\}$. The policy evaluation logic determines whether the request should be granted based on applicable bindings and inheritance. This logic is implemented in the transition relation. Specifically, for state $s \in S$ and action $a = (m, r, p, res, c)$, the next state $s' = Grant$ if there exists a binding $(m, r', res', c')$ in the policy such that: (1) $r' = r$, (2) $(p, r) \in PA$, (3) $res' \preceq res$, and (4) $c'$ evaluates to true given the request context (or $c' = \perp$). Otherwise, $s' = Deny$. The initial state $s_0 = Deny$, representing default denial.

Security requirements are expressed as temporal logic properties. We use Computation Tree Logic (CTL), which allows quantification over execution paths. The basic property pattern is the response pattern: if a certain condition holds, then eventually a response should occur. For access control, this becomes: if an access request with specific characteristics occurs, then eventually a grant (or deny) decision should be made. Formally, $\forall\Box(condition \rightarrow \forall \diamond decision)$, where $\Box$ means "always", $\diamond$ means "eventually", and $\forall$ quantifies over all paths.

In NuSMV syntax, CTL formulas use different notation: $AG$ for $\forall\Box$, $AF$ for $\forall\diamond$, and $A$ for the universal path quantifier. Thus, the response pattern becomes $AG(condition \rightarrow AF(decision))$. The condition includes predicates on request parameters: member, role, permission, resource, and condition. For example, to verify that member "alice@example.com" should never be granted the "storage.objects.delete" permission on resource "bucket-1", we would check $AG((member = \text{"}alice@example.com\text{"}) \wedge (permission = \text{"}storage.objects.delete\text{"}) \wedge (resource = \text{"}bucket - 1\text{"}) \rightarrow AF(decision = Deny))$.

To start with, The model, or the policies, is a transition system. Rule in transition relation corresponds to a binding. The relationship between a resource and a manager assigns bindings relevant to communication with the resource and impact. The conditions are encoded as a Boolean expression which can be evaluated on the request context. Next, the security properties are formalized as CTL formulas. The organisational requirements shall govern the formulae. Thirdly, the model checker verifies each property by exploring the whole state space. When there is a property in the model then the verification is succeeded. In addition, if verification fails, the model checker produces a counterexample. A counterexample consists of a sequence of states and actions that violate the property.

Counterexamples are very helpful in correcting policy mistakes. A counterexample where verification fails shows values of the requests that causes the result to differ from the property's requirement. As an illustration, let us say a property states that a user must not have a delete permission on a bucket. If the verification is unsuccessful, then the counterexample displays it to the user, permission and resource. With this information, an administrator may repair the policy (for example, by removing a binding or by

adding a more restrictive condition) [16].

The approach allows both positive verification checking, ensuring you can access what you want, and negative verification checking, ensuring you cannot access what you do not want. Positive verification involves the verification on a positive use case denoting that it does allow the legitimate users to do whatever they need to do. It aids in business operation. Negative verification determines if your organization is adhering to least privilege and separation of duties. Do both things.

Scalability considerations are very important to any practical application. Google Cloud organizations can contain multiple thousands of resources, roles, and users for example. Exhaustive modeling of every entity can hence cause state explosion. We use abstraction techniques in our implementation to counter complexity. An illustration is similar resources are grouped together or resource members and permissions are represented symbolically. The NuSMV model checker uses symbolic algorithms to efficiently manage large state spaces (BDD-based). In large instances the verification is done in parts of the policy using modular verification.

One clear benefit of our approach is that it follows NIST guidance on the verification of access control. NIST recommends the utilization of formal methods for high-assurance systems following a large-scale literature review. By automating verification of AC, it minimizes human verification effort. The human verification of complex systems has less computational power than intelligence assignment and is therefore slow under many constraints.

## 7. Case Studies and Verification Results

We evaluate our verification framework through three case studies derived from Google Cloud IAM documentation. Each case represents a realistic scenario involving different GCP services, resource types, and security requirements. For each case, we model the policies, formulate verification properties, and analyze the results. All models and properties are implemented in NuSMV and available for replication [17].

### 7.1 Case 1: Cloud Pub/Sub Messaging

The first case involves Cloud Pub/Sub, a messaging service. The scenario includes two resources: project "project-a" and topic "topic-a" within that project. Two users exist: "bob@gmail.com" and "alice@gmail.com". A policy assigns the "roles/pubsub.editor" role to bob@gmail.com on project-a. Another policy assigns the "roles/pubsub.publisher" role to alice@gmail.com on topic-a. Due to inheritance, topic-a inherits the policy from its parent project-a, so bob@gmail.com also has editor role on topic-a.

We model this scenario with three bindings: (1) bob@gmail.com as pubsub.editor on project-a, (2) alice@gmail.com as pubsub.publisher on topic-a, and (3) bob@gmail.com as pubsub.editor on topic-a (inherited). The transition system encodes these bindings and the inheritance relationship. We then verify several properties.

First, we verify that alice@gmail.com has publish permission on topic-a: $AG((member = \text{"}alice@gmail.com\text{"}) \wedge (permission = \text{"}pubsub.topics.publish\text{"}) \wedge (resource = \text{"}topic-a\text{"}) \rightarrow AF(decision = Grant))$. This property holds, confirming intended access. Second, we verify that alice@gmail.com does not have publish permission on project-a: $AG((member = \text{"}alice@gmail.com\text{"}) \wedge (permission = \text{"}pubsub.topics.publish\text{"}) \wedge (resource = \text{"}project-a\text{"}) \rightarrow AF(decision = Deny))$. This also holds, as her role is bound only to topic-a, not project-a.

Table 3: Case 1: Cloud Pub/Sub Verification Results

| **Test** | **CTL Formula —> Result** | **CEx** |
|---|---|---|
| Pub access | $AG(A \wedge p \wedge t \rightarrow AF(Grant))$ | No |
| No pub on proj | $AG(A \wedge p \wedge pr \rightarrow AF(Denyed))$ | No |
| Inheritance | $AG(B \wedge t \rightarrow AF(Grant))$ | No |
| No delete | $AG(A \wedge d \wedge t \rightarrow AF(Denyed))$ | No |
| Pub error | $AG(A \wedge p \wedge pr \rightarrow AF(Grant))$ | Yes |

Table 4: Case 2: Cloud Storage Verification Results

| **Test Case** | **CTL Formula ——-> Result** | **CEx** |
|---|---|---|
| Group create | $AG\big((\text{g} \wedge \text{c} \wedge \text{b}) \rightarrow AF(dec = \text{Grant})\big)$ | No |
| Group no delete | $AG\big((\text{g} \wedge \text{d} \wedge \text{b}) \rightarrow AF(dec = \text{Deny})\big)$ | No |
| Alice delete | $AG\big((\text{A} \wedge \text{d} \wedge \text{b}) \rightarrow AF(dec = \text{Grant})\big)$ | No |
| No org delete | $AG\big((\text{any} \wedge \text{d} \wedge \text{org}) \rightarrow AF(dec = \text{Deny})\big)$ | No |

Third, we check inheritance: bob@gmail.com should have editor permissions on both project-a and topic-a. The property

$$AG\big((\text{member} = \text{"bob@gmail.com"}) \wedge (\text{resource} = \text{"topic-a"}) \rightarrow AF(\text{decision} = \text{Grant})\big)$$

The verification successfully confirms expected behavior and would detect errors if present. For instance, if an administrator mistakenly added a binding granting alice@gmail.com publisher role on project-a, the second property would fail, producing a counterexample showing the violation. This demonstrates how verification can catch unintended permission assignments.

## 7.2 Case 2: Cloud Storage Bucket

The second case involves Cloud Storage, with a bucket "upload-here" in project "project-a". Two user types exist: a data processing expert "alice@example.com" and a group "data-uploaders@example.com" containing three members. Policies assign "roles/storage.objectAdmin" to alice@example.com on project-a, and "roles/storage.objectCreator" to the group on project-a. The bucket inherits these policies. The security requirement is that group members can upload objects but not delete them, while alice@example.com can both upload and delete [18].

We model this with bindings for alice@example.com and each group member, with inheritance to the bucket. Properties verify that: (1) group members have create permission on the bucket, (2) group members lack delete permission on the bucket, (3) alice@example.com has both create and delete permissions on the bucket, and (4) no one has delete permission on the organization node "example.com" (higher in hierarchy).

All properties hold as expected. For example, the property $AG((member = \text{group-member}) \wedge (permission = "storage.objects.delete") \wedge (resource = "upload-here") \rightarrow AF(decision = Deny))$ is true because the objectCreator role does not include delete permission. The property $AG((member = "alice@example.com") \wedge (permission = "storage.objects.delete") \wedge (resource = "example.com") \rightarrow AF(decision = Deny))$ is true because her role is bound at project level, not organization level.

Table 5: Case 3: Compute Engine Results

| Test | CTL Formula —> Pass | CEx |
|---|---|---|
| Bob net inst-a | $AG(B \wedge n \wedge ia \rightarrow AF(G))$ | No |
| Alice inst-b | $AG(A \wedge i \wedge ib \rightarrow AF(G))$ | No |
| Alice ¬inst-a | $AG(A \wedge i \wedge ia \rightarrow AF(D))$ | No |
| Alice ¬proj1 | $AG(A \wedge a \wedge p1 \rightarrow AF(D))$ | No |
| Alice proj1* | $AG(A \wedge a \wedge p1 \rightarrow AF(G))$ | Yes |

This case demonstrates verification of least privilege: group members have only the permissions necessary for their task (uploading), not excessive permissions (deleting). It also shows hierarchical boundary verification: permissions granted at project level do not incorrectly propagate to organization level.

### 7.3 Case 3: Compute Engine Instances

The third case involves Compute Engine virtual machines. The organization "example.com" contains two projects, each with a VM instance. User "bob@example.com" has network admin role at organization level, inherited to both projects and their instances. User "alice@example.com" has instance admin role on project-2 only, inherited to its instance but not project-1's instance. This tests inheritance across sibling branches in the hierarchy [19, 20].

We model bindings accordingly and verify properties about cross-branch isolation. Key properties include: (1) bob@example.com has network admin permissions on both instances (inheritance across branches), (2) alice@example.com has instance admin permissions on instance-b but not instance-a, (3) alice@example.com lacks instance admin permissions on project-1.

All verification succeeds. For instance, $AG((member = "alice@example.com") \wedge (permission = "compute.instances.create") \wedge (resource = "instance - a") \rightarrow AF(decision = Deny))$ holds because her role is bound to project-2 branch only. Similarly, $AG((member = "bob@example.com") \wedge (resource = "instance - b") \rightarrow AF(decision = Grant))$ holds for network admin permissions due to organization-level binding.

This case highlights verification of cross-branch isolation, an important security concern in multi-project environments. It ensures that roles bound to one branch do not unintentionally affect sibling branches. The model checker confirms that inheritance flows downward within a branch but not across branches unless explicitly bound at a common ancestor.

### 7.4 Utility of Counterexamples in Policy Remediation

When a security property fails verification, the NuSMV model checker produces a counterexample—a concrete trace showing the sequence of states and actions that led to the violation. In the context of IAM, such a counterexample reveals:

- the member (user or service account) making the request,
- the resource being accessed,
- the permission requested,
- the role binding(s) that granted the access,

- the inheritance path (e.g., organization → folder → project) that propagated the permission.

This information enables administrators to pinpoint the exact binding responsible for an unintended privilege. For instance, a counterexample might show that a user inherited a delete permission from an organization-level role that was intended only for project-level use. Corrective actions can then be taken such as removing the binding, adding a condition, or refining the role definition without having to manually audit the entire policy set. The availability of precise counterexamples transforms verification from a mere compliance check into an actionable remediation tool.

### 7.5 Analysis and Discussion

In every case, verification confirmed that the behavior was correct and would be flagged otherwise. The documented examples should be correct, which is to say, there shouldn't be any true errors in them. In a real deployment on a real system, a similar verification would catch a misconfiguration. Generating counterexamples is a particularly useful ability as they give.

The verification method is sufficiently scalable for real-world situations. Every case model has 5-10 bindings, 3-5 resources, as well as various users of a small to medium policy set. Verification can be done per project or folder, independently of others, ensuring the use of a modular approach. NuSMV employs symbolic model checking techniques using BDDs to enhance state space management.

It is important to note that the IAM policies of Google must be manually translated to the formal model. Translating automatically would enhance our tool, although experts in the field consider it to be easy. As our future work, we are planning a compiler from the google's policy language to the NuSMV input. Our work also has limitations related to dynamic conditions that depend on the context of executing the request, e.g, time-of-day. It is assumed by our model conditions.

Despite these shortcomings in some scenarios, it remains capable of providing enormous security. When a policy eventually gets deployed, it is formally verified at high level by the cloud administrator and you have strong assurance that deployment will not cause a data breach and non-compliance. A secure verification tool may also provide support for audit, attestation or compliance certification. So long as there is no need for program assistance, this can be quite beneficial.

## 8. Conclusion and Future Work

This paper presented a formal verification framework for Google Cloud IAM policies. We developed $\text{RBAC}_{\text{GCP}}$, We offer a detailed formal model for Google IAM's RBAC which has resource hierarchy and conditional bindings. We implemented the model as a transition system, which allows model-checking of IAM(Identity and Access Management) of Google with NuSMV system. The tool's security properties are expressed as temporal logic formulas that allow the automatic verification of authorization, prohibition, inheritance, and least privilege properties.

The framework was clearly illustrated with three different case studies. In every instance, verification was successful as the documented policies demonstrate the expected behaviour. The first example from the Cloud Pub/Sub service demonstrates correct allowing access. the Cloud Storage service provides the next example of correct "deny" access. The firewall policy manifests correctly in our last Compute Engine example. Since none of the documented cases contain any error, our strategy has failed to produce any counterexample for them. The verification will usually fail.

## References

[1] D. Jain. Assortativity in k-nearest neighbor (k-nn) graphs for high-dimensional datasets, 2025.

[2] D. Jain. Evaluating traditional machine learning models for environmental sound classification, 2025.

[3] D. Jain. Enhancing human motion analysis with deep learning-based wearable IMU systems, 2025.

[4] Deepit Sapru. Analyzing crime discourse in U.S. metropolitan communities on Reddit: Trends, influences, and insights, 2025.

[5] Deepit Sapru. GeoSense-AI: Fast location inference from crisis microblogs, 2025.

[6] Dhruv Dixit. A contrastive meta-learning approach with isotropic sparse decomposition for scalable audio-visual learning. In *[Missing Conference Name]*, pages –, 2025. doi: 10.1007/978-3-031-79041-6_21. Full conference name not provided in the original citation.

[7] Oyeronke Ladapo. Revolutionizing data preparation and access for visual and multi-modal business analytics. *International Journal of Science and Advanced Technology*, 16(2), 2025. doi: 10.71097/IJSAT.v16.i2.3490.

[8] Oyeronke Ladapo. Dynamic self-adaptation in server systems for optimized performance and availability. *International Journal of Science and Advanced Technology*, 16(2), 2025. doi: 10.71097/IJSAT.v16.i2.3487.

[9] Oyeronke Ladapo. Empowering energy efficiency: A real-time mobile analytics platform for intelligent consumption monitoring. *International Journal of Science and Advanced Technology*, 16 (2), 2025. doi: 10.71097/IJSAT.v16.i2.3486.

[10] Siddhant Sukhatankar. Adaptive layer calibration: Performance boost for large models. Technical report, TechRxiv, December 2025.

[11] Mridul Banik. Novel tensor norm optimization for neural network training acceleration. In *Proceedings of the 2025 International Conference on Artificial Intelligence and Its Applications (icARTi '25)*, pages –, 2025. doi: 10.1145/3774791.3774805.

[12] Siddhant Sukhatankar. Behavioral threat unmasking: A system defense paper. Technical report, TechRxiv, December 2025.

[13] Siddhant Sukhatankar. Visualizing optimization feedback: Latent space analysis embedding visualization. Technical report, TechRxiv, November 2025.

[14] Mridul Banik. LLM tuning: Neural language persistence through adaptive mixture. In *Proceedings of the 2025 International Conference on Artificial Intelligence and Its Applications (icARTi '25)*, pages –, 2025. doi: 10.1145/3774791.3774803.

[15] Mridul Banik. Instructional video summarization with transformers: A curriculum learning approach, 2025.

[16] E. Zahoor, A. Ikram, S. Akhtar, and O. Perrin. A formal approach for the identification of authorization policy conflicts within multi-cloud environments. *Journal of Grid Computing*, 20(2): 18, 2022.

[17] A. Gouglidis, I. Mavridis, and V. C. Hu. Security policy verification for multi-domains in cloud systems. *International Journal of Information Security*, 13(2):97–111, 2014.

[18] A. Sissodiya, E. Chiquito, U. Bodin, and J. Kristiansson. Formal verification for preventing misconfigured access policies in kubernetes clusters. *IEEE Access*, 2025.

[19] H. Benattia. *Formal Modelling and Verification of Security Policies in Cloud Computing*. PhD thesis, Université Mohamed Khider-Biskra, 2019.

[20] M. Drouineaud, M. Bortin, P. Torrini, and K. Sohr. A first step towards formal verification of security policy properties for RBAC. In *Fourth International Conference on Quality Software, 2004. QSIC 2004. Proceedings*, pages 60–67. IEEE, 2004.