

1 Server-Side Data Access Control

Access control determines which users are permitted to access a server's resources and which operations they may perform. In a data-oriented service, users may be allowed to query a database, retrieve particular records, execute specific computations, or access a machine learning model, depending on their identities, roles, attributes, and subscription plans.

For example, a service may allow free users to execute a limited number of queries per day while premium users can execute more queries or access additional datasets. Similarly, a research platform may permit users to query public data while restricting sensitive data to authorized researchers.

The objective of access control is to ensure that every request is evaluated against an explicitly defined access policy before the requested operation is performed.

1.1 Authentication, Authorization, and Enforcement

Server-side access control involves three distinct operations:

- **Authentication:** Verifies the identity of the requester, for example, using passwords, multi-factor authentication, or cryptographic credentials.
- **Authorization:** Determines whether the authenticated user is permitted to perform the requested operation on a resource under the applicable policy.
- **Enforcement:** Ensures that the operation is executed only if authorization succeeds. Enforcement may occur at an API gateway, application server, database, or multiple layers.

Authentication alone does not imply authorization. A successfully authenticated user may still be denied access to a particular dataset or query operation.

1.2 Access-Control Model

Let $\mathcal{U}$ denote the set of users, $\mathcal{R}$ the set of resources, and $\mathcal{O}$ the set of operations. An operation may include reading a record, executing a query, modifying data, or invoking a model inference service.

An access-control policy can be represented as a decision function

$$\text{Authz} : \mathcal{U} \times \mathcal{R} \times \mathcal{O} \times \mathcal{C} \longrightarrow \{\text{Allow}, \text{Deny}\},$$

where $\mathcal{C}$ represents contextual information, such as subscription status, remaining query quota, time, or project membership.

For a request (u, r, o, c) , the server evaluates

$$d = \text{Authz}(u, r, o, c).$$

The requested operation is executed only if $d = \text{Allow}$. Otherwise, the request is rejected.

A secure implementation should deny requests by default and explicitly grant only the permissions required by each user.

1.3 Access-Control Policies and Subscription Tiers

A common implementation uses Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), or a combination of both.

- **RBAC:** Permissions are assigned to roles, and users obtain permissions through their assigned roles. Examples include free user, premium user, and administrator.
- **ABAC:** Access decisions depend on attributes of the user, resource, requested operation, and environment. Examples include subscription status, dataset sensitivity, project membership, and request time.

Consider a service that offers free and premium subscriptions. Let $\text{tier}(u)$ denote the user's subscription tier, $\text{active}(u)$ indicate whether the subscription is valid, and $\text{quota}(u)$ denote the user's remaining query quota. A possible policy is

$$\begin{aligned} \text{Authz}(u, r, \text{query}, c) = & \text{Authenticated}(u) \\ & \wedge \text{active}(u) \\ & \wedge (\text{quota}(u) > 0) \\ & \wedge \text{Permitted}(u, r). \end{aligned}$$

Here, $\text{Permitted}(u, r)$ checks whether the user's subscription and other attributes authorize access to resource r . The policy can be extended to impose different quotas or resource permissions on free and premium users.

For example, premium users may be granted access to additional datasets, higher query quotas, or more computationally expensive operations. The server must verify the current subscription entitlement rather than relying solely on a client-supplied claim.

Query quotas also require enforcement of concurrent requests and atomic quota updates to prevent users from exceeding their limits through parallel requests. Subscription expiration, cancellation, and permission changes must be reflected in subsequent authorization decisions.

1.4 Enforcing Access to Server Data

A typical access-control workflow is

$$\text{Request} \rightarrow \text{Authentication} \rightarrow \text{Authorization} \rightarrow \text{Query Execution} \rightarrow \text{Result}.$$

Authorization should be enforced on every relevant request at multiple levels:

- **API-level:** Controls access to endpoints and operations.
- **Query-level:** Restricts permitted queries and computations.
- **Data-level:** Limits access to specific tables, records, or fields.
- **Usage-level:** Enforces quotas, rate limits, and subscription restrictions.

These controls are complementary. API access does not imply unrestricted data access; the server must also validate queries and enforce data-level permissions to prevent unauthorized access.

2 Is Cryptography Required for Access Control?

Cryptography is useful for access control, but it is not necessarily required to implement the access-control policy itself. The choice depends on the trust assumptions and security requirements of the system.

Conventional server-side access control. When the server is trusted to enforce policies, access control can be implemented using authentication, authorization logic, secure session management, and database permissions. Encryption in transit, such as TLS, protects requests and responses against network interception, while encryption at rest protects stored data against certain forms of storage compromise.

However, encryption at rest does not prevent an authorized application server from accessing the plaintext. If the server decrypts the data and executes queries on it, the server must enforce the access policy correctly.

Cryptographic credentials and tokens. Digital signatures and message authentication codes (MACs) can protect the integrity and authenticity of credentials, tokens, and authorization-related messages. For example, a server may issue a signed token containing a user's identity and subscription claims. A receiving service can verify the signature before accepting those claims.

Nevertheless, a valid signature does not by itself establish that a subscription is still active or that the requested operation is authorized. The system must also validate the claims, their validity period, revocation status where applicable, and the current access policy.

Cryptographic enforcement for untrusted storage. Suppose the server or cloud storage provider is not trusted to enforce access restrictions. In that setting, conventional application-level authorization may be insufficient because the provider could bypass the policy and read the stored data.

Encryption-based access control can reduce this dependence on the storage provider. Data can be encrypted before outsourcing, and decryption keys can be distributed only to authorized users.

Attribute-Based Encryption (ABE) is one possible mechanism. In ciphertext-policy ABE, an encryption policy is associated with a ciphertext, and a user can decrypt it only if their cryptographic attributes satisfy the policy, subject to the security guarantees of the construction. For example, a policy may require both membership in a research project and an active authorization attribute.

ABE introduces additional challenges, including key management, attribute revocation, collusion resistance, and policy updates. Moreover, ordinary ABE does not automatically support arbitrary queries over encrypted data. Searchable encryption or other specialized cryptographic mechanisms may be needed when the server must search or compute over encrypted data without learning the underlying plaintext.

3 Access Control for AI/ML Services

Access control in AI/ML systems determines who can access training datasets, invoke models, inspect model parameters, or use specific inference features. For example, free users may have limited query quotas, while premium users receive higher quotas or access to additional models. Sensitive datasets may be restricted to authorized researchers.

Such policies are enforced through authentication, authorization, database permissions, and usage controls. Cryptography can protect communications and credentials, or enforce access restrictions when storage is untrusted. However, restricting model access does not prevent information leakage through authorized outputs; techniques such as differential privacy may be needed to limit such leakage.