network

MDPI

Article

Alohomora: Workflow-Aware Authentication and Authorization in Heterogeneous Systems

Hussain M. J. Almohri

Department of Computer Science, Kuwait University, Kuwait City, Kuwait; almohri@cs.ku.edu.kw

Abstract

Current federated identity management systems lack contextual awareness of workflows across independent systems, creating security gaps and workflow integrity challenges. This article details the design and implementation of Alohomora, a distributed workflow-aware authentication system that maintains cross-system workflow context through path-bound tokens. Alohomora complements existing identity providers such as OAuth and SAML by adding workflow orchestration capabilities while leveraging standard authentication protocols for initial user verification. The system introduces workflow graphs as a formal model for representing dependencies between functions across heterogeneous systems and employs a distributed caching architecture with collaboration groups for scalable session management. In a typical deployment scenario, an employee onboarding workflow across human resources services, account provisioning, and benefits systems forms a trust group where Alohomora enforces ordered step execution, validates prerequisite completion at each transition, and generates cryptographic completion assertions upon workflow finalization. Extensive performance evaluation under concurrent user requests demonstrates polynomial performance characteristics with superior scalability compared to centralized OAuth introspection. The results show that Alohomora maintains high throughput under heavy load while providing strong, secure access control through workflow path binding and distributed trust orchestration. The prototype implementation is available as open source.

Keywords: authentication; authorization; distributed systems; cross-system authentication; session management; distributed caching

check for updates

Academic Editor: Marco Savi

Received: 5 October 2025
Revised: 30 October 2025
Accepted: 2 November 2025
Published: 5 November 2025

Citation: Almohri, H.M.J. Alohomora: Workflow-Aware Authentication and Authorization in Heterogeneous Systems. *Network* **2025**, *5*, 51. https://doi.org/10.3390/network5040051

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Web application software needs authentication and authorization to secure and monitor access to its data. Web application developers have two main choices that they share with their user base. A user can often choose to create a credential set (typically a username and a password) or to use a credential set registered with another system, either from the same organization or an independent entity. The first choice is suitable when the application software wants to maintain its own authentication process and data with a potential performance gain. The second choice works well for offloading the burden of authentication on a more specialized service that continuously develops the authentication software, improving integrity and availability. Because the authentication software itself is the source of revenue for the provider, they are motivated to apply best practices, improve the protocol, and provide a high-quality service. These are advantages gained by using software as a service in general, which have contributed to the growth of web application software in various domains.

When sharing identity management, Single Sign-On (SSO) is a ubiquitous choice. Single Sign-On is an authentication mechanism that allows users to securely access multiple independent systems using a *single* credential set through centralized authentication, typically managed by a trusted identity provider (IdP). SSO improves user experience as it reduces the number of authentications needed as a user navigates between two systems owned by the same organization sharing a common credentials database.

Federated Identity Management (FIM) goes beyond the capabilities of SSO by decentralizing authentication across multiple independent organizations. FIM allows a single authentication from one system to be used in another independent system through standardized protocols like OAuth [1], OpenID Connect [2], or SAML [3]. Each organization manages its own identity provider (IdP), enabling cross-domain authentication without centralized credential storage. While improving scalability, interoperability, and user convenience across different administrative domains, FIM introduces complexity in trust establishment, policy alignment, and federation agreements among participating entities and demands rigorous security controls to manage interorganizational risks.

Although SSO and FIM are widely used in practice, the state of the art for federated identity management has a subtle but important gap: the lack of maintaining, controlling, and enforcing workflow integrity. When using FIM across many independent web application services belonging to various organizations, the systems can work interdependently to execute a workflow. For example, a user logs into a system in an organization and starts executing a workflow. The workflow depends on the completion of steps in a second system in another organization or entity.

The problem with the core design of FIM is the lack of contextual awareness. The user uses FIM to log into the second system and continue the workflow. In current FIM models, the second system is unaware of access control rights and intentions of the user. After completion of the workflow, FIM or SSO is silent about a systematic way to provide proof of the completion of the workflow steps for the user. The problem occurs because FIM is not designed with contextual workflows in mind, as the protocol focuses solely on secure authentication that provides an authentication proof token to other independent systems. Thus, the authentication token does not maintain the broader context of the workflow, which triggers the authentication process.

Existing solutions complement federated identity management or provide an alternative. Security Assertion Markup Language (SAML) is one of such models, which is an open standard for exchanging authentication and authorization data between parties, primarily between an identity provider (IdP) and a service provider (SP). SAML uses Extensible Markup Language (XML) to exchange contextual information such as assertions about an authenticated user and the user’s permissions. Although SAML provides an important component of federated identity management, it lacks a mechanism to follow and present the proof of task completion in a workflow environment. Another method of multiple system authentication is through a central system that would use Attribute-Based Access Control [4] to associate the access control context with system functions. A central system inherently only works with services that belong to the same legal identity or organization and suffer from scalability problems.

This article presents Alohomora , a workflow authentication orchestration and monitoring system, which enhances federated identity management. The name *Alohomora* is inspired by a charm that is used to unlock doors [5] in Harry Potter stories. Alohomora leverages the new model, workflow graphs, which provides provable and verifiable workflow context (user identity, role, and workflow progress assertions) in addition to federated identity management and authentication. Alohomora is primarily concerned with systems that belong to multiple independent organizations (such as government agencies), implementing complex and interdependent workflow steps. Alohomora does not replace federated identity management models such as OpenID or OAuth. Instead, Alohomora provides the data structures and protocols to ensure the proper and secure execution of workflow processes.

Alohomora also provides workflow completion assertions, an important component that is absent from existing federated identity management solutions. An assertion is a cryptographic signature by Alohomora's main server, showing the execution of an instance of a workflow, the user that completed the steps, and the servers that provided the service. The assertion can be queried through Alohomora's server or could be inserted into the token T for a serverless verification of workflow completion.

To illustrate Alohomora's functions, consider an employee onboarding workflow (detailed in Section 2.4) across multiple systems forming a trust group. When an employee authenticates with the HR system, Alohomora registers a path-bound token encoding the complete workflow dependency graph. As the employee transitions between systems (account provisioning, benefits enrollment), each system validates the token with Alohomora, verifying both user identity and completion of prerequisite workflow steps. The system enforces ordering by rejecting access to workflow steps whose dependencies remain incomplete, while maintaining workflow state throughout the process. Upon completing all steps, Alohomora generates completion assertions for systems participating in a workflow.

To prevent a single point of failure, Alohomora can enable participating workflow servers to act as authentication session caches. A replica server joins a group of service providers and continuously synchronizes a cache of authentication tokens and system information.

This article presents the following contributions.

1. A distributed workflow-aware authentication system that enforces ordered step execution through path-bound tokens, preventing unauthorized cross-system access while maintaining workflow context across heterogeneous systems (Section 3).
2. A formal workflow graph model encoding multi-system dependencies with cryptographic completion assertions that enable verifiable proof of workflow execution (Section 3.3.3).
3. A distributed replica caching architecture demonstrating substantial cache hit rates and reduced query latency compared to centralized validation (Section 5.5.4).
4. Performance evaluation across geographically distributed deployments demonstrating scalable concurrent user support with comparable throughput to centralized OAuth introspection (Sections 5.5.3 and 5.4).
5. Security analysis proving that workflow path constraints restrict token validity to dependency-defined sequences, preventing privilege escalation and cross-workflow token misuse (Section 4).

2. Background and Related Works

2.1. Authentication, Authorization, and Trust

Authentication is the process of proving the identity of a registered user with a specific software system S. The authentication process establishes a temporary proof that an interaction with a system's function $f \in F(S)$ under an identity A has indeed been initiated by the owner of the identity A. The proof is based on the identity A providing a previously recorded secret (password).

OpenID Connect (OIDC) is an identity authentication that helps with distributed authentication [6]. OpenID Connect standardizes authentication by providing protocols for creating and querying about *authentication tokens*. It complements authorization frameworks by adding an identity layer that allows applications to verify user identity and obtain basic profile information through standardized ID tokens in JavaScript Object Notation Web Token (JWT) format.

Authorization is another critical integrity requirement. Authorization is a process that (independent of authentication) verifies and enforces mandatory access control on a subset of the system's functions $F^*(S) \subseteq F(S)$. A discrimination of user privileges is a necessary building block to allow a reliable interoperability between several systems. Thus, we assume that, to use distributed authentication and authorization, a system must reliably define fine-grained authorization privileges for its users that can be propagated and translated to other systems in the distributed environment.

OAuth 2.0 is a standard authorization framework that works with OpenID Connect. With OAuth, users assert application privileges, for example, enabling access to some of their data hosted by a particular application [7]. OAuth 2.0 provides access tokens for API authorization, whereas OpenID Connect provides both access tokens and ID tokens that contain authenticated user identity claims. There are applications where OAuth, similar to OIDC, can be used for authentication as well. Users can use their Google account credentials to log into other applications that use Google as the identity provider (IdP). OAuth is a complex ecosystem that has been shown to be vulnerable to attacks, especially due to loose security measures by the identity providers [8]. Similarly, OpenID's complexity has led to serious attacks [9].

2.2. Heterogeneous System Authentication Approaches

Various approaches exist to authenticate users in multiple heterogeneous systems with functional dependencies (summarized in Table 1). We give an account of the related work in the context of central authentication databases, federated authentication, and distributed and decentralized authentication.

Table 1. Comparison of Alohomora with Existing Authentication Solutions. The columns capture architectural styles (centralized vs. distributed design), availability of token binding (mandatory access control), step ordering (mandatory sequential workflow execution), completion proofs (validating prerequisite step completion), replica caching (distributed state caching support), cross-org (support for cross-organizational federation), and workflow context (maintains multi-system workflow state).

System	Architecture	Token Binding	Step Ordering	Completion Proofs	Replica Caching	Cross-Org	Workflow Context
Kerberos	Centralized	User-bound	No	No	Limited	Limited	No
OAuth 2.0	Centralized	User & scope	No	No	No	Yes	No
SAML	Federated	User & service	No	No	No	Yes	No
OpenID Connect	Centralized	User & scope	No	No	No	Yes	No
Ledgers	Distributed	User-bound	No	Ledger	Varies	Yes	No
IDaaS	Centralized	User-bound	No	No	Vendor-dep	Vendor-dep.	No
Alohomora	**Distributed**	**Path-bound**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

Central Databases. A distributed authentication system can be designed and implemented using several approaches. The simplest approach is to designate a central database of user profiles and their access rights. In this design, the owner of the central database registers a set $\mathcal{S}$ of participating systems that can query the central database for user authentication and authorization. All systems in $\mathcal{S}$ enjoy the same level of access to the central database. One can model access rights to the central database in a two-ring system. The most privileged ring is the owner that can add and delete a system S_i to the set $\mathcal{S}$. All systems in $\mathcal{S}$ are horizontally assigned query rights.

Kerberos [10], a distributed authentication system, depends on a central database that is managed within a single organization (realm). Although the authentication function itself is distributed in Kerberos *partially*, multiple systems request authentication, but one database verifies the user. Thus, Kerberos faces a scalability problem when expanding to a large system base across multiple organizations. As a result, the authentication function cannot be readily available to multiple organizations.

Authentication in web applications is also problematic in Kerberos. Common Web server software such as Apache 2.0 HTTP Server [11] can be configured to request an authentication ticket and verify a user's identity [12]. However, this function works when the user logs into the web application from within an intranet. OAuth, a single sign-on [13] authentication standard (or SPRESSO, a privacy-enhanced variation [14]), solves this issue by providing identity as a service but itself suffers from various security weaknesses [15].

Identity as a Service (IDaaS) is another centralized authentication method [16]. Using IDaaS, a system delegates identity management to a cloud provider. The objective of the service is to provide the software tools and storage required for identity management in a centralized location. IDaaS providers are usually designed to work with a specific vendor (such as Oracle's IDSC [17]), making them inoperable across multiple platforms and thus inapplicable for heterogeneous systems.

Federated Identity Management. Federating the authentication process is a possible solution to use IDaaS in multiple organizations. In this approach, a cloud identity provider can interact with a localized identity management service that is managed by another organization. As a result, a system S can consult the IDaaS service or its own identity management service for authentication. The federation component is enabled using the Security Assertion Markup Language (SAML) [18,19], which enables the exchange of sensitive authentication data between two identity management partners. SAML is a step forward for providing decentralized, federated, multi-organizational authentication using a set of protocols for authentication in heterogeneous environments.

Decentralized and Distributed Ledgers. As an alternative to centralized databases, systems can use distributed ledgers for authentication [20,21]. A ledger $\mathcal{L}$ is a chain of tamper-proof transactions that can be appended and verified independently, usually with a consensus algorithm [22,23]. No central authority can control or disable access to the ledger. Each system can act as a verifier and maintain a full copy of the ledger. When a user presents their identity, the system consults $\mathcal{L}$ to obtain a verified and most recent password for the identity and then decides whether the user should be authenticated.

There are challenges facing distributed and decentralized ledgers. First, authentication requires solid verification of identity when the user registers in the system. This requires user provisioning in which a (central) entity provides original *evidence* [24] for the identity to establish trust. Second, the distributed nature of the ledger requires continuous synchronization and redundancy, which can result in large ledgers with a time-consuming identity search. Third, revocation of permissions can be challenging. Should the system add a new block to the ledger indicating that the identity is no longer valid, or should it search through every single copy of the ledger and delete a user's block?

2.3. Context-Aware Identity Management

An emerging alternative to the prominent existing solutions is context-aware authentication and authorization systems. Contextual information is a necessity when implementing authentication in distributed cloud environments [25]. Li et al. present the concept of context-aware authorization sequences [26]. The system addresses OAuth's limitation of allowing arbitrary order and unlimited use of permissions within validity periods.

The distributed authentication design presented in this article, referred to as Alohomora, complements existing identity management systems by adding workflow context to authentication and authorization processes. Alohomora uses a central system but also provides the opportunity to employ identity caches, reducing the processing burden on a central system. It also provides strict authorization verification by maintaining formal workflows, employing the concept of collaborative workflow groups, and using hybrid client-side and server-side session tokens. Alohomora's design draws inspiration from several established security models but includes novel elements specific to cross-system authentication workflows. The path-bound token approach shares conceptual similarities with capability-based security systems, where access rights are unforgeable and tightly scoped. Similar to Role-Based Access Control with delegation, Alohomora enables controlled propagation of authentication context while maintaining separation of duties. However, unlike these existing models, Alohomora specifically addresses the challenge of authentication in heterogeneous distributed systems with functional dependencies.

Table 1 highlights critical differences in workflow support. Existing systems lack mechanisms to enforce ordered workflow execution or validate step completion across heterogeneous systems. OAuth and OpenID Connect provide stateless tokens with no workflow context. SAML assertions are point-in-time claims without cross-system state propagation. Alohomora introduces path-bound tokens that restrict authentication validity to specific workflow sequences and include completion information for validation. The system can, in principle, complement existing identity providers by adding workflow orchestration while leveraging standard authentication protocols for initial user verification. However, the technical details of integrating Alohomora with existing systems is left for future work.

The core technical difference between Alohomora and traditional Federated Identity Management (FIM) systems lies in the path-bound token implementation mechanism. Unlike FIM bearer tokens that grant access based solely on scope claims, Alohomora tokens embed a workflow identifier that references a formal dependency graph encoding the complete workflow structure. During validation, the system verifies not only token authenticity but also that the requesting system participates in the bound workflow path and that prerequisite workflow steps have been marked complete in the centralized state. This stateful validation enables enforcement of execution ordering across system boundaries, a capability absent in stateless FIM protocols. Furthermore, Alohomora maintains distributed replica caches synchronized with the main server, allowing systems to validate workflow state locally, while FIM systems require round-trip introspection to the identity provider for each validation.

2.4. Motivation for Designing Alohomora

A typical use case starts with a new employee authentication with the registration system (S_1). Next, the human resources (HR) system (S_2) must poll S_1 or receive notifications to know when the registration is complete. After processing the user request in S_2, the employee receives separate instructions to complete three different tasks.

The employee must navigate and authenticate with each of the three systems (S_3, S_4, S_5) independently. Each system maintains its own session and progress tracking without awareness of the other systems.

An option to improve the user's experience is to employ a central database for authenticating users. This creates a coupling with the central authentication system and can cause availability problems when the authentication service is unavailable. The administration of workflow creates additional coordination challenges that require an efficient and secure orchestration framework.

Using the workflow-aware authentication system with path-bound tokens, the systems behave in a more efficient way. Consider a use case in an employee onboarding system. The use case starts with a new employee authentication with the registration system (S_1). An additional step is introduced, requiring S_1 to create a path-bound token encoding the complete workflow path and transfers the user to S_2. In turn, S_2 receives a token and validation of the workflow from Alohomora and automatically authenticates the employee for HR processing. After HR processing, S_2 updates the token with completion status and presents the employee with the three parallel tasks. The employee can visit S_3, S_4, and S_5 in any order, with each system automatically authenticating the user using the same path-bound token. Each system updates the token with its completion status while maintaining the workflow context.

Throughout the process, the employee maintains a single authentication session with the full workflow state preserved in the path-bound token. No custom integrations or polling mechanisms are required between systems; the workflow state travels with the token. If the process is interrupted, the employee can resume from any point with the complete workflow context maintained.

The scenario discussed shows two areas for improving existing solutions. The first is the reduction in authentication steps required in complex workflow-based multi-system processes, which is a time-consuming usability burden. The second is the lack of explicit cross-system workflow integrity that, if complemented by identity management systems, would enable secure context-aware authentication among collaborating but independent software systems.

3. Design of Alohomora

3.1. Design Goals

Based on the limitations of existing solutions and the requirements of cross-system authentication workflows, Alohomora conforms to the following goals.

Workflow-bound Authentication Sessions. Alohomora implements workflow-bound user sessions that restrict authentication token validity to specific workflow paths in an environment where multiple independent systems (such as microservices, which require careful considerations [27]) collaborate to accomplish a specific business workflow. Unlike generic tokens used in traditional single sign-on (SSO), these path-bound tokens can only be used for the precise system functions defined in a workflow dependency graph (defined in Section 3.3.3). Section 4 provides a thorough security analysis, showing that path-bound verifiable tokens provide the necessary context to prevent misusing authenticated session tokens.

Managed Trust Relationships. Authentication session sharing implies *indirect* transitive trust between collaborative systems with functional dependencies. Alohomora acts as a trusted orchestration layer, maintaining authentication integrity while allowing systems to remain logically independent. Section 4 demonstrates that Alohomora's assertions prevent session hijacking and impersonation attacks.

Scalability and Reliability. Despite acting as a centralized trust layer, Alohomora maintains performance by using service providers as potential session caches. When a service provider uses a session, the session is inevitably stored in the local database used by the service. Alohomora provides a unified application programming interface (API) that can be exposed by the providers and used by other servers to query the validity of tokens, lowering the query burden on Alohomora's main server. Section 5 presents a thorough performance analysis of a prototype implementation.

True single sign on. Alohomora implements true single sign-on as it automatically enables sharing a single user session among systems that have functional dependencies. Users authenticate once and seamlessly transition between systems in a workflow without repeated authentication prompts. Alohomora's design demonstrates that even in complex scenarios where systems interact in multistep workflows, Alohomora requires a single authentication session to execute a workflow.

3.2. Design Overview

Alohomora implements a distributed authentication architecture that orchestrates workflow-bound sessions across independent systems. Figure 1 illustrates the key components and their interactions in a typical cross-system authentication workflow.

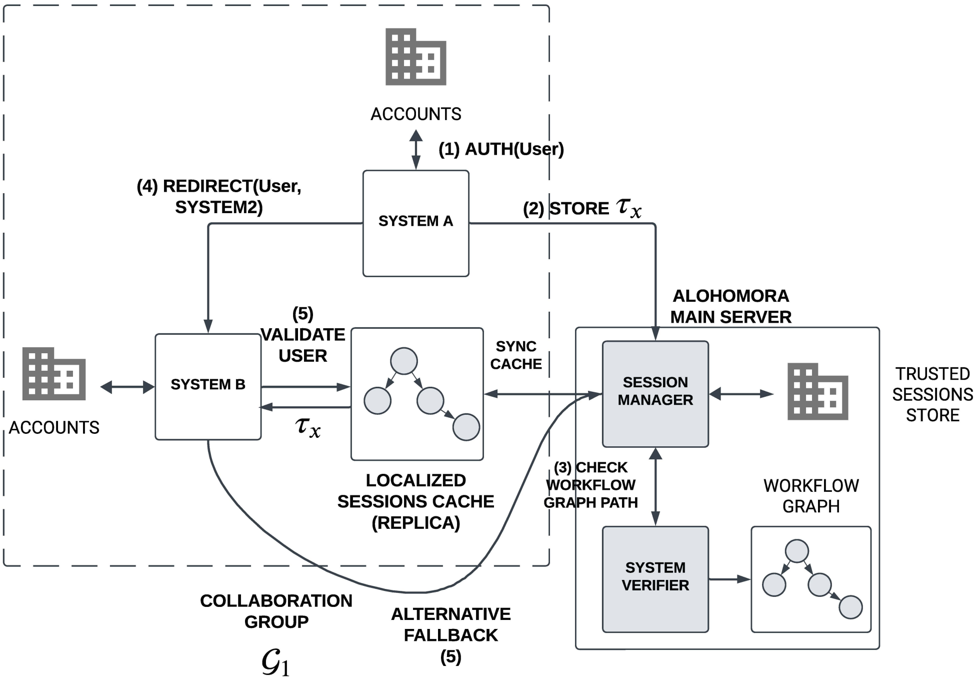

Figure 1. Alohomora Architecture Overview. The diagram shows the core components and authentication flow: (1) the user authenticates with System A, (2) System A registers path-bound token τ_X with the main server, (3) the main server synchronizes updates to collaboration group caches, (4) the user accesses System B with token τ_X, (5) System B validates the session via the main server or cache, and theuthorized access granted. The dashed box indicates the collaboration group $\mathcal{G}_1$.

As depicted in Figure 1, the architecture comprises four key components. First, *subscribed systems* (such as Systems A and B) provide services to users after authentication. Second, *workflow graphs* ($G = (V, E)$) define authorized paths $(u, v) \in E$ between system functions V, capturing the collaboration among multiple subscribed systems. Third, *Alohomora's main server* maintains the authoritative session state and maintains the workflow. Through path-bound tokens, the main server restricts the authentication context to specific workflow sequences, ensuring that compromised tokens cannot be misused across unauthorized system boundaries. Lastly, *replica caches* provide distributed session cache storage and provide session validation on behalf of the main server, reducing the centralized session-validation burden.

3.2.1. System Components

A *subscribed system* $S_i \in \mathcal{S}$ is software that is registered and authorized to use Alohomora's services (Section 3.3.1). When S_i authenticates a registered user account u, it produces an *authenticated user session* α_u, asserting that S_i has authenticated the user u with valid credentials. Alohomora maintains the session database store, the system registry, the API servers that provide endpoints for session operations, and the cache synchronization service that maintains consistency between distributed cache nodes, enabling scalable session verification endpoints.

The workflow dependency digraph must accurately represent the authorized paths between system functions, and all communications between subscribed systems and Alohomora must use secure channels (e.g., transport layer security). Although systems may trust Alohomora with session management, they maintain sovereignty over their internal authorization decisions.

3.2.2. Threat Model and Adversary Capabilities

Network Sniffing. In the threat model, several types of adversaries are considered. An adversary on the network can observe, intercept, and modify network traffic between user agents and systems. These adversaries search for Alohomora-owned artifacts to subvert authentication and authorization procedures.

Session Hijacking. Adversaries attempt session hijacking and steal session information and replay captured authentication tokens to impersonate a user and benefit from their privileges or violate the user's data integrity (e.g., changing user settings or data stored on a web service). Alohomora does not use browser or application cookies. As detailed later, Alohomora uses two-sided session tokens. The client-side token is a one-time sufficiently large random value that is signed and communicated within encrypted channels. Section 4 thoroughly analyzes the design and its potential to prevent effective session hijacking.

Phishing and user-side compromise. An adversary may attempt to compromise a user's credentials using phishing or other social engineering attacks. Alohomora does not protect against a pure phishing attack in which the user's session has been compromised. However, Alohomora enables surveillance and proactive user notifications. That is, users establish verified communication channels (such as an email or native application notifications) to receive workflow step updates. Although these measures do not eliminate the attack, they help contain the damage from compromised credentials.

System Dependability. Alohomora's main server is secured and mad ereliable using strict standard security solutions such as virtualization-based security [28], hardware-enforced security [29], diversification [30], and fuzzing [31]. This assumption is the building block for assuming that Alohomora's protocols and functionality are reliable.

A registered system (single-sign-on service provider [13]) must maintain the integrity of its user accounts. This assumption is required to maintain the credibility of identity proofs. The trust relationship between two systems depends on a trust group defined later in Section 3.3.4. Specifically, a *subscribed system* $S_i \in \mathcal{S}$ is software that is authorized to use Alohomora's services. When S_i authenticates a registered user account u, it produces an *authenticated user session* α_u, asserting that S_i trusts u. The system S_i's authentication of u indicates that any other system S_j that participates with S_i in the same trust group would trust the authentication result of S_i as performed through the Alohomora protocols defined in Section 3.3.5.

Although Alohomora allows for the transfer of trust within a group, no arbitrary trust relationship is allowed within the Alohomora framework, making it resilient to session hijacking attacks that enable access to any system within the framework. The trust in user sessions within a group is further restricted to the participation of collaborative systems along a specific workflow path. That is, a user session is only valid for use in a certain dependency order on a subset of systems in a group (for further details, see Section 3.3.4). Thus, the assumption is that users may have the intention to *misuse* an authenticated session and attempt to escalate their privileges through accessing system functions for which they do not have permissions. Even under this assumption, the security analysis (Section 4) shows that the session of users will be highly limited and there will be virtually no room for misuse.

3.3. Alohomora Components

For a complete distributed authentication framework, the design incorporates (i) subscribed systems, providing services to arbitrary users over the Internet; (ii) cross-system workflows, which are a formal graph-theoretic model of workflow steps that encompass multiple systems; (iii) trust groups that establish multi-system trust and enable a single authentication session to be accepted across multiple systems; and (iv) cryptographically signed user sessions comprising client-side and server-side components.

3.3.1. Subscribed Systems

As previously defined in Section 3.2.2, a subscribed system $S_i \in \mathcal{S}$ is a complete and independently developed software system. Alohomora registers a subscribed system through a multifactor organizational verification process and provides a cryptographically verifiable system identity $I(S_i)$ (a standard practice in distributed authentication [32]). This identity attestation is time-bounded and includes a digital signature for validation using Alohomora's asymmetric key pair. Formally, $I(S_i) = (\ell, b_i, t, x)$, where ℓ is the system's network endpoint identifier, b_i is the unique system identifier, t is the validity timestamp, and x is the Alohomora-signed message authentication code, which can be implemented using state-of-the-art techniques such as Poly1305-AES [33].

A subscribed system provides an Internet service, usually as web-based APIs, to arbitrary users who have signed up for their services. Alohomora does not provide direct verification of users or their credentials. A subscribed service in Alohomora implements business workflows (Section 3.3.3) that belong to a group of systems (Section 3.3.4). The users in a group are responsible for verifying their trust relationships. Once agreed to form a group with other systems, a subscribed system declares that it trusts user authentication sessions (Section 3.3.5) produced by any of the members of the peer group.

3.3.2. Shared User Tokens

A token is a core instrument of Alohomora. Tokens are provided as proof of authentication to a registered system, implementing a subset of a workflow in a group. Tokens are temporal and also include workflow completion information. Token validation (Algorithm 1) is performed only through Alohomora or a replica and ensures access to workflow steps. Formally, each client-side token τ_X contains the tuple $\langle s, w_{id}, t, n, t_{exp}, m \rangle$, where s is the session identifier, w_{id} is the workflow identifier referencing a registered workflow graph (Section 3.3.2), t is the creation timestamp, n is a cryptographic nonce, t_{exp} is the expiration time, and $m = \text{Poly1305-AES}(k, s \parallel w_{id} \parallel t \parallel n)$ is the message authentication code ensuring token integrity.

3.3.3. Enforcing Path-Bound Tokens

Alohomora's core contribution is path binding through tokens, which is a managed access control mechanism. Path-binding occurs throughout the execution of a cross-system business workflow involving multiple collaborative subscribed systems. Workflows allow tokens to be bound to specific systems and workflow execution steps, limiting token misuse. Alternative access control mechanisms, such as cryptographic access controls, exist. For example, a path could be embedded in the MAC computation for cryptographic tamper-evidence. However, they have been shown to incur prohibitive performance penalties [34]. Further, the mechanism uses workflow identifier indirection to allow path updates without invalidating active tokens.

Mandatory path-binding (Algorithm 1) requires the definition of cross-system workflows. A cross-system workflow represents a sequence of functional dependencies that span multiple independent systems, where completion of one function may be a prerequisite to access another. These workflows emerge naturally in enterprise environments where systems evolve independently but must collaborate to support business processes. Define a *workflow dependency digraph* $G = (V, E)$, where V is the set of vertices corresponding to system functions, and E is the set of directed edges such that $(u, v) \in E$ indicates that the function u depends on the previous completion of function v. The vertex set $V = V_1 \cup V_2 \cup \ldots \cup V_n$ is partitioned into n disjoint subsets, where each subset V_i contains the functions exposed by the system $S_i \in \mathcal{S}$ in this Alohomora instance A.

Algorithm 1 Path-Bound Token Validation. Where $\mathcal{S}_g$ is systems in group g, $\mathcal{T}_u$ is user tokens, $\mathcal{W}_{accessible}$ is accessible workflows, V_{S_i} is functions of system S_i, P_w is workflow path, ρ_τ is session metadata, t_{now} is current timestamp.

```
Require: Requesting system $S_i \in \mathcal{S}$, user identity $u$, collaboration group $g \in \mathcal{G}$
Verify $S_i$'s signature to protect against impersonation.
Validate $S_i$ is registered: $S_i \in \mathcal{S}_g$ for group $g$
if $S_i \notin \mathcal{S}_g$ then
    return UNAUTHORIZED
end if
$\mathcal{T}_u \leftarrow \{\tau \in T : \tau.user\_id = u \wedge \tau.expires\_at > t_{now}\}$    ▷ Valid user tokens
$\mathcal{W}_{accessible} \leftarrow \emptyset$    ▷ Workflows accessible to requesting system
for all $\tau \in \mathcal{T}_u$ do
    Verify $\tau$'s nonce has not been used before
    $w \leftarrow \tau.workflow\_id$    ▷ Token's workflow
    if $\exists v \in V_{S_i} : v \in P_w$ then    ▷ System participates in workflow path
        $\mathcal{W}_{accessible} \leftarrow \mathcal{W}_{accessible} \cup \{w\}$
        $\rho_\tau \leftarrow \{\tau.created\_at, \tau.expires\_at, \tau.metadata\}$    ▷ Collect session metadata
    end if
end for
if $\mathcal{W}_{accessible} = \emptyset$ then
    return NO_VALID_SESSIONS
else
    return $\langle \mathcal{W}_{accessible}, \{\rho_\tau : \tau \in \mathcal{T}_u \wedge \tau.workflow\_id \in \mathcal{W}_{accessible}\}\rangle$
end if
```

For binding of authentication sessions to specific workflows, *directed workflow paths* provide the means for mandatory access control. A workflow path P in a workflow dependency digraph $G = (V, E)$ is a sequence of vertices $\langle v_1, v_2, \ldots, v_n \rangle$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < n$. Each vertex $v_i \in V_j$ belongs to some system $S_j \in \mathcal{S}$. A subset of systems $\mathcal{S}' \subseteq \mathcal{S}$ can share a workflow path P if for each $v_i \in P$, $v_i \in V_j$ for some $S_j \in \mathcal{S}'$. The workflow path P defines the permissible sequence of function invocations across systems that the authentication context may follow.

The workflow dependency digraph defines a partial order over function executions, where the directed edges encode precedence constraints. The graph structure naturally supports fork/join patterns through multiple outgoing edges (fork) and multiple incoming edges (join). For alternative workflow paths, the graph encodes all possible branches, with path-binding restricting tokens to a single selected branch at runtime. To ensure well-formed workflows, Alohomora executes a depth-first search cycle detection algorithm during workflow registration. The algorithm marks vertices as visited during traversal and rejects any workflow graph containing back edges, which indicate cycles. Path-binding enforcement occurs through Algorithm 1, which verifies system membership in the workflow path.

3.3.4. Trust Groups and Relationship

Alohomora provides natural mandatory access boundaries among registered systems as participants of a collaborative group. For example, a system cannot use or receive an access token intended for workflows in which it does not participate.

Continuing the essential components of Alohomora, define a trust relationship $G_{ij} \subset G$ as a subgraph of the workflow dependency digraph that encodes all permitted function dependency paths between systems S_i and S_j. A valid trust relationship must contain at least one cross-system edge connecting a function in S_i to a function in S_j, forming a workflow path P as defined above.

Edges may cross system boundaries, connecting vertices of different subsets V_i and V_j where $i \neq j$. When such a cross-system edge (u, v) exists where $u \in V_i, v \in V_j, i \neq j$, Alohomora recognizes that a trust relationship has been established between systems S_i and S_j. This trust relationship enables S_i and S_j to exchange user authentication sessions along the edge-defined workflow path, with Alohomora mediating the secure transfer of the authentication context.

When creating and validating authentication sessions, the Alohomora instance A enforces path constraints by rejecting any session transfer attempt that does not include a path defined within an established trust relationship G_{ij}. This constraint ensures that the authentication context can only flow along administrator-approved workflow paths. The strict mandatory workflow-bound access sessions are enforced through Alohomora's token generation and validation protocols, as described later in the next section (Algorithm 1).

3.3.5. User Authentication Sessions

A user is represented by authentication credentials (typically a username and password) that allow access to system functions. Each subscribed system S_i performs complete authentication of the users requesting access to protected functions. All functions requiring authentication must be registered as vertices in the system component of the workflow dependency digraph G_i.

Upon successful user authentication, the system creates a workflow-bound authentication session X. Each session is constrained by both temporal limitations and workflow path boundaries, restricting its use to a specific subset of subscribed systems and their exposed functions along authorized paths in the trust relationship G_{ij}. The authentication process produces two distinct security components. A client-side session token τ_X is delivered to the user agent and a server-side session record ρ_X is created by the authenticating system S_i and registered with the Alohomora instance A.

When a subsequent system S_j receives a function access request containing a session token τ_X, it queries Alohomora to verify the validity of the token against the registered session record ρ_X. Alohomora validates that τ_X corresponds to an active session X and that the requested transition from S_i to S_j conforms to an authorized path within the trust relationship G_{ij}.

Alohomora imposes several access control restrictions on user authentication sessions. First, authentication sessions do not automatically transfer trusts. As a mediator, Alohomora only provides a response when a query from a system S_j is received concerning the validation of a token τ_X that is defined on a workflow path P, which includes a function from S_j. Second, Alohomora only allows a system to participate in a workflow if all other systems formally approve the participation through a response to a broadcast request. Alohomora facilitates the approval process to ensure that workflows cannot be compromised by unauthorized systems. Third, Alohomora requires workflow completion assertions from participating systems. That is, suppose that S_1 initiated a user session, marking the completion of a step f_1 on a workflow path P, redirecting the user to S_2. Suppose that

S_2 provides the function f_2 on P, which is a prerequisite to a third function f_3 by a third system S_3. The system S_3 cannot validate the user's session unless f_2 has been marked as complete.

These automatic access control restrictions provide a secure and reliable environment, enabling Alohomora to work as an access control orchestration layer, ensuring smooth and secure transfer of trust among a group of systems.

3.4. Alohomora Protocols

Alohomora defines an initial authentication protocol that extends a typical user's authentication procedure to include the initialization of an Alohomora token session, a token validation protocol for a system on a workflow path to validate if a user has initiated the correct session with Alohomora, and a cache management protocol to enable a distributed cache for improving Alohomora's overall protocol execution performance.

For a system that uses the protocols in an Alohomora instance, it has to be registered, as shown below.

```
Initialize: $\mathcal{S} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset, \mathcal{W} \leftarrow \emptyset$
function RegisterWorkflow($s_{id}, G = (V, E)$)
    $w_{id} \leftarrow$ generateUUID()
    $\mathcal{W}[w_{id}] \leftarrow \{s_{id}, G\}$
    return $w_{id}$
end function
```

Here, $\mathcal{S}$ are registered systems, $\mathcal{F}$ is system functions, $\mathcal{T}$ is active sessions, $\mathcal{W}$ is workflow definitions. Once a system is approved as a registered system in Alohomora, it can request to join a group of systems and manage the group's workflows, subject to approval from all existing group member systems.

3.4.1. Initial Authentication Protocol

The initial authentication protocol is the first step to manage workflow-bound authentication sessions across distributed systems. When a user logs into a registered Alohomora system, the initial authentication protocol is executed.

A successful authentication to the registered System A is followed by registering an Alohomora session for the user to be used by systems that collaborate with System A, shown by the procedure below (where ρ_X is the session record for session X, τ_X is client token for session X, and v_i is function vertex).

```
function RegisterToken($X, \tau_X, P_w, v_i, S_i, u$)
    Create $\rho_X$ using $\tau_X, P_w, v_i, S_i$.
    Broadcast to replicas: $\langle X, \tau_X, P_w, v_i, S_i, u \rangle$
end function
```

Once a token is registered, Alohomora broadcasts the token to the replica servers that are relevant to the workflow's group. Recall that a set of servers can form a group and define a workflow that includes the steps to be executed by the group members. Each group can have one or more replica servers to hold valid user sessions. Alohomora allows group members to validate a user's session through the replicas, as discussed in the next section.

3.4.2. Token Validation Protocol

Validating a token against a workflow path ensures that the authentication context flows only along administrator-approved workflow paths, preventing unauthorized access patterns. The token validation process ensures that workflow-bound authentication sessions are only accessible to systems that participate in the authorized paths, implementing path-bound access control for cross-system workflows. The validation Algorithm 1 enforces that tokens can only be accessed by systems S_i that have registered functions F_{S_i} participating in the same workflows W_w as the token's bound workflow path. This prevents cross-workflow token abuse and implements the path-binding security property. Specifically, a token abuse occurs when a user is allowed to use an authentication session with any system within a distributed authentication platform. Alohomora prevents this abuse by binding a token session to a specific workflow. Further, the token cannot violate the workflow steps. Suppose that the user receives a token τ by accessing the function f_1. The user cannot use the token for f_2 unless the system exposing the function f_1 marks the step as complete.

Token validation is a protocol that takes place when a system needs to verify the user's status not originally authenticated by the system itself. The user accesses a service provided by System B. Then, System B uses the token on the user side τ to validate the token through a replica cache (Algorithm 1). With a cache miss, the system tries the main server for a final validation trial. With a successful response from either the cache or the main server, System B allows for the execution of the function f.

An important step is that System B updates the current workflow execution instance in the Alohomora database, stating that the user completed the function f with token τ. This is a necessary step to enforce mandatory restrictions on the user session tokens that are produced and managed by Alohomora. The enforcement is mandatory because when the user moves to execute the next step in an upcoming system, the system requests the same token validation. Alohomora only validates tokens that are stored in its database, adhere to the workflow, and include the completion of prerequisite steps.

3.4.3. Delta Synchronization for Session Caches

A centralized platform can create performance bottlenecks when handling token validation queries from numerous independent systems. When hundreds of systems simultaneously request token validations, even well-provisioned Alohomora API servers could become overwhelmed. In addition, systems with infrequent validation needs might experience increased latency while waiting for busier systems to be served. To address these challenges, Alohomora uses delta updates, inspired by the `rsync` algorithm [35].

First, identify collaboration groups that cluster systems with high workflow interdependence. Second, define a collaboration group $\mathcal{G}_k$ with identifier k as a set of systems such that for any two systems $S_i, S_j \in \mathcal{G}_k$, there exists at least one edge $(u, v) \in E$ where $u \in V_i$ and $v \in V_j$. This ensures that systems with actual workflow dependencies share cache resources. A system may belong to multiple collaboration groups when it participates in diverse workflows spanning different system clusters. Although the evaluation uses this formal definition to establish collaboration groups, system administrators can manually adjust group composition to optimize performance based on deployment-specific patterns.

Each collaboration group is assigned a dedicated cache server that maintains synchronized session data with Alohomora's main server. The cache server stores a local replica of the authentication session records ρ_X using a schema identical to the main Alohomora database. Session validity is maintained through a periodic synchronization protocol that ensures cache consistency while minimizing main-server load. This distributed architecture significantly reduces the validation latency for systems within the same collaboration group while maintaining the security guarantees of path-bound tokens.

The cache synchronization employs delta synchronization, where replicas maintain timestamps of their last successful update and request only incremental changes since that point, reducing network overhead and improving scalability. Algorithm 2 summarizes the cache synchronization mechanism implemented by Alohomora's main server. The algorithm relies on synchronization timestamps for each registered system replica. Also, the synchronization uses cache groups to avoid wasting synchronization time on sessions that are irrelevant to the group. The server only synchronizes the sessions that are valid, relevant to the group, and have not yet been synchronized with the requesting replica.

Algorithm 2 Main Server Cache Synchronization. Where R is set of replicas, $\mathcal{G}_k$ is collaboration group k, T is time domain, $\mathcal{S}_k$ is systems in group k, $\mathcal{F}_k$ is system functions, $\mathcal{W}_k$ is workflows, $\mathcal{E}_k$ is workflow edges, $\mathcal{T}_k$ is tokens, Δ is update package, T_k is tokens for group k.

```
Require: $replica\_id \in R$, $group\_id \in \mathcal{G}_k$, $last\_sync \in T \cup \{\text{NULL}\}$
Verify $replica\_id$'s signature to protect against impersonation.
if $last\_sync = \text{NULL}$ then
    $\mathcal{S}_k \leftarrow$ all systems in collaboration group $\mathcal{G}_k$
    $\mathcal{T}_k \leftarrow$ all tokens for group $\mathcal{G}_k$
else
    $\mathcal{S}_k \leftarrow$ systems updated since $last\_sync$
    $\mathcal{T}_k \leftarrow \{\tau \in T_k : \tau.created\_at > last\_sync\}$
end if
$\mathcal{F}_k \leftarrow$ system functions for $\mathcal{S}_k$
$\mathcal{W}_k \leftarrow$ workflows for collaboration group $\mathcal{G}_k$
$\mathcal{E}_k \leftarrow$ workflow edges for $\mathcal{W}_k$
$\Delta \leftarrow \langle \mathcal{S}_k, \mathcal{F}_k, \mathcal{W}_k, \mathcal{E}_k, \mathcal{T}_k \rangle$
return $\Delta$ with current timestamp $t_{now}$
```

Each replica server is a read-only server that exposes the query endpoints from Alohomora. It maintains a timely, fresh copy of sessions created and stored at Alohomora that are assigned to the group the replica belongs to. Algorithm 3 lays out the steps for synchronizing the sessions with the main Alohomora server. The algorithm periodically sends a synchronization request to the main server. With a successful response, the replica updates its local cache, which uses the exact database schema of the main Alohomora server.

Algorithm 3 Replica Cache Synchronization. Where M is main server, R is set of replicas, $\mathcal{C}_r$ is local cache of replica r, Δ is update package, $base\_interval$ is minimum retry delay, and max_delay is maximum retry delay.

```
Require: Main server $M$, replica $r \in R$, collaboration group $\mathcal{G}_k$
Verify $M$'s signature to protect against impersonation.
$\mathcal{C}_r \leftarrow \emptyset$                                        ▷ Initialize local cache
$failure\_count \leftarrow 0$                                ▷ Track consecutive sync failures
while Service active do
    $\Delta \leftarrow$ Request updates from $M$ for group $\mathcal{G}_k$
    if $\Delta \neq \emptyset$ then
        Merge records.
        $failure\_count \leftarrow 0$
    else
        $failure\_count \leftarrow failure\_count + 1$
        $delay \leftarrow \min(base\_interval \times 2^{failure\_count}, max\_delay)$
        Wait for $delay$ before next attempt
    end if
    Wait for synchronization interval
end while
```

The delta cache synchronization mechanism in Alohomora employs several efficiency factors. First, Alohomora uses two-way synchronization. That is, the main server pushes token updates in batches when the server is under relatively lower load. Alternatively, replicas issue periodic database pull requests using Algorithm 3. Alohomora's push updates are not required if the replica is capable of executing the pull updates. However, these two mechanisms ensure that replicas are kept up-to-date even under occasional delays. Second, when Alohomora is not responsive, replica synchronization fails (Algorithm 3). In this case, the replica implements an exponential backoff [36], giving the main server more time to recover from its failure or reduce its load before responding to the replica synchronization update request. Third, as shown in Algorithm 2, delta updates only provide the replica with tokens that belong to their group, have fresh information (new tokens or updated tokens) and have not expired.

Alohomora employs adaptive eventual consistency for cache synchronization (Algorithms 2 and 3). Each replica maintains a replica-specific synchronization interval that adapts to workload: when cache miss rates increase, the system accelerates synchronization frequency; as misses decrease, the interval relaxes toward a configured minimum, balancing staleness with synchronization overhead.

3.5. Token Management

This section specifies operational mechanisms for token management in distributed deployments, addressing clock synchronization in Section 3.5.1, key lifecycle in Section 3.5.2, token duration control in Section 3.5.3, and handling cache failures in Section 3.5.4.

3.5.1. Clock Skew Mitigation

Since multi-step workflows require substantial completion times dominated by user interaction rather than system processing, Alohomora is able to use moderate tolerance to allow for synchronization gaps. For example, in a multi-step employee onboarding system, the employee needs to register, complete forms, take surveys, or complete multiple additional steps. Thus, the cooperating systems may require token lifetimes extended for several hours.

Alohomora provides two mechanisms for clock synchronization that benefit from intrinsic features of the framework. First, workflow-bound tokens naturally tolerate clock skew through extended lifetimes. In this case, token expiration is set as $t_{exp} = t + \alpha \cdot T_w$ where safety factor $\alpha \in [2, 3]$ accounts for delays and T_w is the mean completion time for the workflow. With tokens valid for hours, clock drift on the order of minutes becomes negligible.

Additionally, subscribed systems synchronize with their collaboration group replicas every $\Delta t_{sync} = 300$ s. Each system sends timestamp probe $\langle t_i \rangle$ and receives $\langle t_r, \Delta t \rangle$ where $\Delta t = t_r - t_i$ is the observed drift. If $|\Delta t| > 300$ s, the system adjusts its local offset. This hierarchical synchronization approach, distributes load across replicas rather than centralizing at the main server. Temporal validation accepts tokens when $|t_{now} - t| < \min(300\text{s}, t_{exp} - t)$.

3.5.2. Key Rotation Policy

Each collaboration group establishes a key expiration policy for member systems. When a system establishes a new group (becomes group owner), it sets the first expiration policy. Other systems joining the group can suggest new expiration policy that needs consensus from the group members. A key expiration policy specifies the maximum lifetime of a key ℓ_g given a group g. When system S_i's key age exceeds ℓ_g, Alohomora issues renewal notifications, managed by the group's replica server. Systems must register with fresh cryptographic keys through the standard registration protocol (Section 3.3.1) before reaching ℓ_g.

Failure to renew results in automatic decommissioning. That is, the system is removed from active collaboration group membership and cannot validate new tokens. Existing in-flight tokens remain valid until natural expiration to avoid disrupting active workflows. This approach, analogous to X.509 certificate lifecycle management [37], prevents indefinite use of potentially compromised keys while minimizing operational disruption. Upon successful registration, the system rejoins with minimal downtime.

3.5.3. Duration Control

Token duration is application-specific. In Alohomora, groups can specify duration control as global parameters, which will be enforced by the replica or the main Alohomora server (as they synchronize group data). When defining a workflow graph $G = (V, E)$, the system specifies expected completion time T_g as metadata. Token expiration is computed as $t_{exp} = t_{create} + \alpha \cdot T_g$ where $\alpha \geq 2$ provides a safety margin. Validation (Algorithm 1) enforces $t_{now} < t_{exp}$ at every step. Expired tokens are rejected regardless of cryptographic validity. This decentralized approach allows heterogeneous duration policies within a single collaboration group without per-workflow server configuration.

3.5.4. Handling Cache Failure

When the replica cache lookup fails, systems employ a three-tier fallback. First, a subscribed system can query alternate group replicas. The Alohomora model allows for multiple replicas to be registered. Systems can either use replicas horizontally or vertically. That is, when a replica fails or a request to the replica is timed out, the subscribed system can try the next replica. In a horizontal replica access, systems employ a round-robin schedule to access the available replicas. The next fallback option is to issue token lookup requests to the main server with exponential backoff. Finally, the subscribed system can reject the user's login request using a cumulative timeout (with HTTP 503) or initialize the login locally. To minimize cold-start misses, Alohomora proactively pre-populates replicas when new workflows are registered. For long-idle workflows receiving first tokens after hours, the main server pushes tokens to all group replicas immediately rather than waiting for periodic synchronization.

3.5.5. Workflow Modification and Token Lifecycle

When a workflow definition is modified, the system distinguishes between active tokens bound to the previous workflow version and new tokens using the updated definition. Active tokens remain valid until expiration, allowing in-flight workflows to complete under their original definitions, avoiding mid-execution disruption. For long-running workflows spanning days or weeks, tokens maintain extended expiration times configured through duration control (Section 3.5.3), with the system tolerating clock skew through the mechanisms described in Section 3.5.1. Explicit revocation is achieved by setting token expiration to the current timestamp, causing Algorithm 1 to reject the token. This approach ensures that both the main server and replicas honor revocation without requiring distributed coordination beyond the existing synchronization protocol.

3.5.6. Group Management and Security Guardrails

Group membership creation follows a consensus-based approval process: when a system requests to join a collaboration group, all existing group members must approve the request before Alohomora registers the new member. The group owner (the system that established the group) audits membership and workflow definitions through the main server's administrative interface. However, new group membership or changes to workflow steps require the consensus of the entire group. To prevent misconfiguration, Alohomora enforces automatic security checks during workflow registration: the cycle detection algorithm (Section 3.3.3) rejects graphs containing circular dependencies, and the system validates that all referenced system identifiers correspond to registered group members. Additionally, the path-binding validation in Algorithm 1 provides runtime guardrails by rejecting tokens for workflows in which the requesting system does not participate, preventing unauthorized cross-workflow access even if workflow definitions contain errors.

4. Security Analysis of Alohomora

This section provides a comprehensive security analysis of Alohomora's distributed authentication architecture, examining its resilience against common security threats, and comparing its security posture with existing authentication systems.

4.1. System Integrity

As mentioned in Section 3, each system participating in Alohomora must be registered and given a cryptographically signed system identity $I(S_i)$. The system identity is used in all message exchanges between a registered system and Alohomora's main server or the replica server. When registering a system, Alohomora's operator is assumed to verify the identity of the system's owner. This identity verification is necessary when Alohomora is installed to be used for a specific set of systems, such as a group of government enterprises.

Alohomora's authentication endpoints do not depend on the integrity of the registered systems. Thus, Alohomora's endpoint security takes standard measures for protecting a system's surface, such as preventing denial-of service attacks, input sanitation, and remote intrusion prevention. These are security measures that are beyond the scope of this article.

To analyze the integrity of Alohomora's authentication logic, consider two attack scenarios. Suppose that Alohomora has three registered systems, S_1, S_2, and S_3, with an agreed-upon workflow w (submitted to Alohomora and approved by all participating systems) with the workflow edge $u_1 \rightarrow u_2 \rightarrow u_3$, where u_i is a function registered in system S_i.

In the first attack scenario, an unregistered malicious user a exploits a vulnerability at S_1 and tricks it to authenticate a, producing a valid Alohomora session token τ for a. The user's motivation is, for example, to gain government approval after executing the workflow w without registration (bypassing legal approvals). With no defense mechanism, the malicious user a can move on to execute the rest of the workflow at S_2 and S_3 as they would trust S_1 with the initial authentication.

Alohomora uses two mechanisms to reduce the impact of the unregistered user attack. First, because of workflow-awareness, Alohomora sends push notifications, informing participating systems about the completion of workflow. In this case, S_1 receives the completion notification for a user that has been authenticated but not registered in the system, causing a conflict. Second, Alohomora employs a random second authentication for a workflow. Alohomora forces a second authentication by randomly deactivating tokens prematurely. This requires a simple database update query, which is automatically synchronized with replicas using Algorithm 3. Random token deactivation forces systems S_2 and S_3 in the scenario above to authenticate the user, reducing the probability that the attack is successful. The probability is proportional to the number of systems on the workflow path and is dependent on the integrity of each participating system.

In the second attack scenario, an unregistered malicious user impersonates a registered user to gain a session token throughout the workflow path w. Alohomora only mitigates this attack using the random mandatory authentication. Although the random mandatory authentication can theoretically be subverted, the attacker needs to exploit *all* systems on the workflow w to ensure that the attack is always successful. Thus, as mentioned earlier, as the number of systems increase in w, the probability of the attacker's success declines.

4.2. Session Security

Alohomora maintains client-side τ_X and server-side ρ_X session data. No client can prove possession of a valid session if any of the two parts of a session is missing. A client without τ_X cannot prove that it has successfully registered in session X. A fabricated session token without a corresponding Alohomora registered token ρ_X cannot provide proof of valid authentication.

Alohomora implements path-bound token security through workflow constraints embedded within each token structure. Unlike traditional bearer tokens that provide broad system access, workflow-bound tokens contain cryptographically protected path specifications that restrict their usage to predetermined system sequences. When a token τ_w is generated for workflow path $w = \langle S_i \rightarrow S_j \rightarrow \ldots \rangle$, the path constraints are embedded using the Poly1305-AES message authentication code that prevent modification without invalidating the token. Any attempt to use the token at a system not specified in the workflow path results in automatic rejection during the validation process defined in Algorithm 1.

The token lifecycle has temporal security mechanisms that prevent replay attacks and ensure session freshness. Each token includes a generation timestamp and expiration time that are validated against the current system time during authentication. Additionally, Alohomora employs nonce-based token generation where each token contains a unique identifier that prevents successful replay of intercepted tokens. The combination of temporal validity and unique identifiers creates a security boundary that limits the window of opportunity for potential attackers even if they successfully intercept valid authentication tokens.

4.3. Cache Security

Cache security in Alohomora's distributed architecture addresses the unique challenges of maintaining session integrity across replica servers. Each replica cache maintains session data through secure synchronization protocols that include signature verification for all incoming updates. When the main server broadcasts session updates to replica caches, each update package contains cryptographic signatures that replicas verify before incorporating new session data. This signature-based integrity check prevents cache poisoning attacks in which malicious actors attempt to inject false session information into the distributed cache network.

To transmit tokens between two systems, Alohomora takes advantage of the existing identity infrastructure of the system to create secure communication channels. All intersystem communication occurs through Alohomora-mediated channels rather than direct system-to-system messaging, eliminating potential vulnerabilities in peer-to-peer authentication protocols. When a user transitions from one system to another within a workflow, the authentication context travels through Alohomora's secure infrastructure, ensuring that tokens are never exposed to potentially insecure direct communication paths between subscribed systems.

Although stealing data from a replica is possible if a replica server is compromised, the scope of an effective attack is severely limited due to built-in mechanisms in Alohomora. Suppose a replica server R was previously registered with an Alohomora instance A through the initial system registration procedure. R receives an initial batch of session tokens $T_R \subset T_A$ (where T_A is the set of all tokens in Alohomora). The update occurs while R is in the *benign* state with no malicious control. R is then remotely compromised and transitions to the *malicious* state. From the point of compromise, the rogue replica server R can leak the subset $T_R^* \subseteq T_R$ containing valid tokens that have not expired. In addition, R can request a local cache update from A, keeping the set T_R up to date.

An attacker in control of R can use a valid user's session to perform malicious activities on the user's behalf without the user's knowledge. The attack results in the same outcomes as session hijacking attacks discussed earlier. Three mechanisms limit the scope of the attack. First, the attacker only has access to read-only session data because the cache synchronization protocol has no definition for updates from the replica side. The systems interacting with the replica do not accept any update request from R. The systems send only token validation requests. Second, the set of stolen tokens T_R^* is limited to a specific set of workflows involving the collaboration group to which R belongs. Thus, the attacker has no access to the rest of the A's tokens.

The attack is severely limited due to an inevitable session conflict. Continuing the attack scenario, the attacker retrieves a user a' session τ and attempts to present it to the system S_2 following the completed step of the token S_1 (the token embeds completion information). S_2 validates τ with R, the validation is successful, and the attacker proceeds. The actual user a also presents τ to S_2, which immediately detects a conflict. S_2 does not need to validate τ with R and is certain that the user's session is compromised or the replica R is rogue. S_2 communicates the problem with the main server in Alohomora A, pauses further updates to R, and informs all systems in the group about the possible compromise of the user a. If the incidents continue to occur with many other user sessions (less likely a user-side compromise), Alohomora requires a fresh replica registration, ensuring forensic analysis from the replica operator.

Mitigating a cache misuse attack may indicate a potential denial of service vulnerability, which arguably is not possible given the attack scenario above. The attacker may plan to use the mitigation mechanism (by suspending synchronization requests to R). The denial of service is nonexistent if the attacker already controls R as a denial-of service attack would deny the attacker fresh token updates. Suppose that the attacker attempts to deny the user service by triggering dual session validation attempts on the system S_2. This attempt requires a replay attack, which should trigger a compromised user flag. Note that a single incident would not pause replica updates since Alohomora should be certain about a replica compromise if a diverse set of users repeatedly sends dual token validation requests. This attack by itself requires the attacker to compromise a large set of users in the collaboration group, which is itself a reason to alert all the systems and stop replica updates.

When replica compromise is detected, Alohomora executes a structured recovery protocol: (1) the main server immediately suspends synchronization to the compromised replica, preventing further token leakage; (2) all systems in the affected collaboration group receive notification to bypass the compromised replica and query the main server directly; (3) the replica operator performs forensic analysis and system remediation; (4) after remediation, the replica undergoes fresh registration with new cryptographic credentials, including key rotation as described in Section 3.5.2; and (5) the main server performs full state synchronization with the recovered replica before restoring it to active service. This recovery process ensures that compromised replicas are isolated, analyzed, and properly restored without disrupting workflow execution for the collaboration group.

5. Evaluation

This section presents a comprehensive evaluation of Alohomora's distributed authentication architecture through controlled experiments and performance analysis. The analysis includes a comparison of Alohomora with OAuth introspection-based authentication in multiple dimensions, including scalability, latency characteristics, and architectural effectiveness.

5.1. Objective, Research Questions, and Metrics

The main objective of the experiments is to show the performance of Alohomora under simulated load (Section 5.3). A comparison of Alohomora's performance with the OAuth Introspection mechanism is presented in Section 5.4. The experiments investigate

1. The system's scalability by measuring throughput under increasing concurrent load,
2. The performance impact of replica architecture in Alohomora, comparing it to direct main server access,
3. Comparing Alohomora against OAuth in terms of response time consistency and tail latency behavior under stress, and
4. Whether token caching provides measurable performance improvements in high-concurrency scenarios.

The experiments are performed in two settings. First, a large-scale simulation on a simulated local machine network demonstrates the scalability of the model and the comparison with OAuth. Second, a network of machines' virtual private clouds is created in three different geographic locations where Alohomora is tested for request processing and cache performance under real network delays.

System performance (throughput) T is measured in users/second, representing the maximum rate at which the system can process concurrent workflow executions as $T = \frac{N}{t}$, where N is the concurrent user count and t is the average pure system processing time that excludes delays in user behavior.

Response Time Percentiles capture latency distribution characteristics. P50 (median) indicates a typical user experience, P95 represents an acceptable worst-case performance for most users, and P99 captures extreme tail latency behavior.

Finally, the replica hit rate measures the percentage of session queries successfully resolved by replica servers: $R_{hit} = \frac{RH}{TQ} \times 100$, where RH is the replica hit count compared to the total queries TQ.

5.2. Experimental Prototype

The experiments analyze Alohomora's performance using a prototype testbed, which is published as an open source project on https://github.com/kussl/Alohomora (access on 30 October 2025). The prototype includes the full implementation of Alohomora's protocols. The experiments use an example Alohomora-enabled employee onboarding system prototype. The example system implements the interactions with Alohomora, but the core functions of the system are not implemented as they are not the focus of the study.

The simulated workflow starts with the user registration, followed by a human resource (HR) processing step, which then redirects the user to execution of a background check, an information technology (IT) account setup, and enrollment in benefits. Each of these steps are designated as an independent Python-based microservice, exposing an API endpoint to conduct the required user activity.

All experiments were conducted on a machine with 6-Core Intel Core i7, running macOS Sequoia. The software stack includes Python 3.9.1, Flask 2.x web framework, SQLite 3.x database, and Python requests library for HTTP communication. Each microservice in the testbed is a Flask application that runs on the Flask built-in HTTP server. The workflow simulator runs each application on a distinct local port and executes JavaScript Object Notation (JSON) requests between any two applications. Alohomora's main server also runs as an independent application and is accessible through the built-in HTTP server in Flask.

All communication occurs over localhost with network latency limited to 1 ms, ensuring that network effects do not dominate performance measurements. The replica synchronization interval is set to two seconds with a 15-s initial sync delay to maximize replica effectiveness.

5.3. Alohomora Performance Analysis

Three experiments (on a local machine) measure the overall system's throughput, workflow completion time, and the cache mechanism's efficiency. Figure 2 reveals the system's scaling characteristics and workflow completion behavior. The figure shows the actual Alohomora's performance with a polynomial fit line. When measuring the system's throughput and workflow completion times, the system receives an increasing simulated system load (50–1000 users). Each user performs actual HTTP requests to the systems that use Alohomora's services, implementing the employee onboarding workflow. Systems in turn use Alohomora's endpoints to store and validate session tokens, while Alohomora also maintains a fresh cache replica server in the system group.

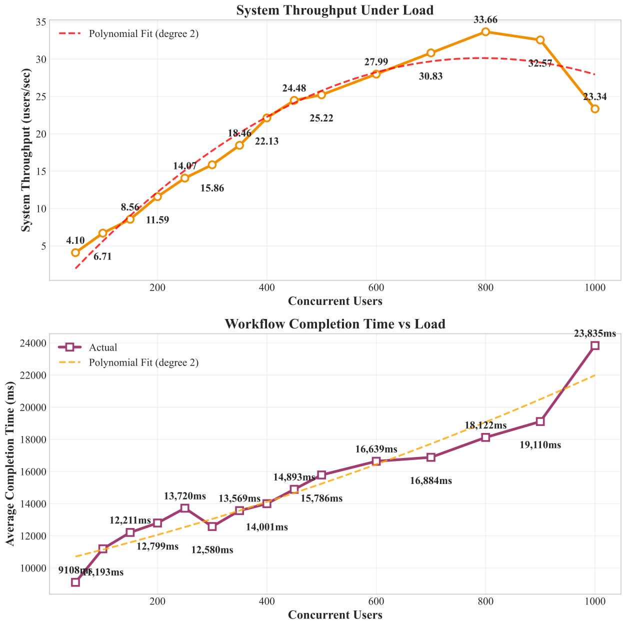

Figure 2. System Throughput and Workflow Completion Time Analysis. The dashed line indicates polynomial fit.

System throughput follows a polynomial curve, peaking at 33.66 users/s at 800 concurrent users before declining to 23.34 users/s at 1000 users. This indicates optimal operating capacity around 700–800 concurrent users. Notice that the simulation testbed poses limitations, such as the maximum number of acceptable connections on the experimental HTTP server and the limited number of cores available on the test machine.

The average workflow completion time scales nearly linearly from 9108 milliseconds (ms) for 50 users to 23,835 ms for 1000 users, representing a 2.6x increase.

The experiments assess the efficiency of the cache mechanism by measuring the number of cache hits compared to main server fallback, along with the average response time from the endpoint (the response either comes from a replica or the main server). Figure 3 demonstrates the critical performance difference between replica hits and main server fallback operations. When session data are successfully retrieved from replica servers, response times remain consistently low (816–1329 ms) on moderate user loads. However, when replica synchronization fails, forcing fallback to the main server, response times increase dramatically (5519–18,047 ms), representing a 3–10x performance penalty. This result suggests the efficacy of the distributed cache mechanism compared to centralized session validation queries.

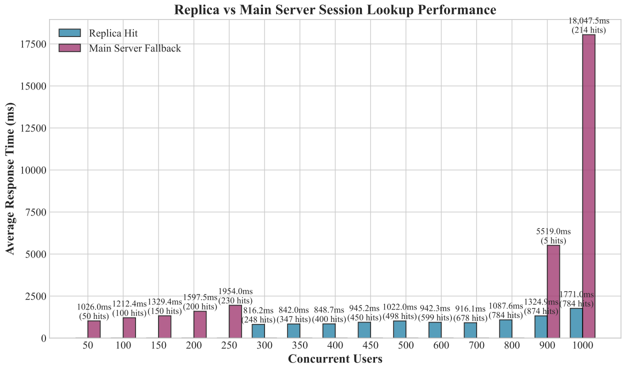

Figure 3. Performance analysis of session lookup response times along with cache hits or misses.

The cache efficiency experiment further reveals distinct operational phases. The replica hit rates remain at 0% for loads up to 250 concurrent users, indicating that replica synchronization mechanisms fail to establish effective data consistency during initial scaling phases. For moderate loads (300–800 users), replica systems achieve near-perfect hit rates (100%), representing the optimal operating window for distributed architecture benefits. At maximum load (1000 users), the replica hit rate drops to 78.6%, suggesting that the synchronization mechanisms become overwhelmed.

It is beneficial to analyze the distribution of response times with an increasing number of concurrent queries to Alohomora when the cache is enabled. As Figure 4 shows, the response times from three major Alohomora protocols show consistent patterns. The response time variations are not significant except for a few outliers that could be under the influence of other system services.

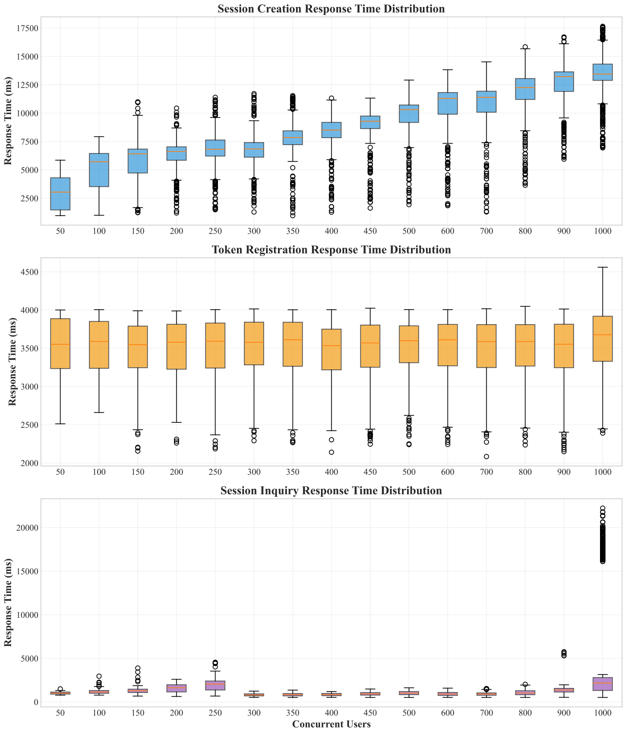

Figure 4. Box plots showing the distributions of query response time with respect to session creation, token registration, and session inquiry.

5.4. Performance Comparison Results

The OAuth Introspection is a centralized model where each participating system holds opaque access tokens and must query a central introspection server to validate token authenticity and extract the user's information. The introspection server maintains the authoritative token state and responds to validation requests with active/inactive status and associated metadata.

The experiments use four comparison approaches: (1) Alohomora with replica support, (2) Alohomora without replica (centralized mode), (3) OAuth Introspection server, and (4) OAuth Introspection with caching. All these experiments are conducted on a local machine.

Scalability Analysis. The throughput analysis (Figure 5) reveals three distinct performance tiers. Alohomora variants demonstrate superior scalability, with standard Alohomora achieving 97 users/s at 1000 concurrent users, while Alohomora (no replica) peaks at 111 users/s at 800 users before stabilizing at 95 users/s. Both introspection variants plateau around 65–69 users/s, with minimal difference between cached and non-cached variants at high load.

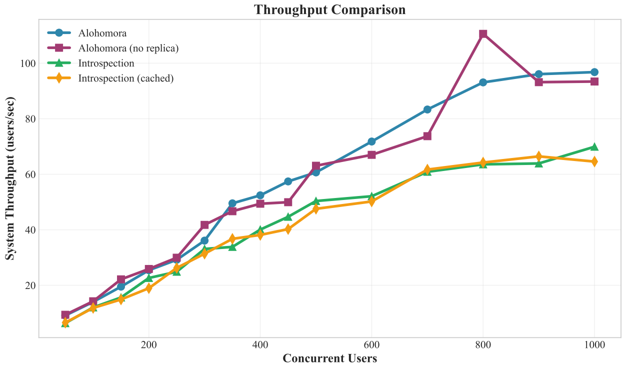

Figure 5. System throughput comparison across the tested variants (Alohomora with and without cache and introspection server with and without cache).

Alohomora (both variants) exhibits consistent linear scaling. This shows an effective load distribution through the replica architecture. Introspection systems show similar scaling patterns. The lack of significant differences between cached and non-cached variants in the introspection server suggests that caching overhead may offset cache benefits under high concurrency.

Replica Architecture Effectiveness. The comparison between Alohomora variants directly addresses replica effectiveness. Alohomora (with replica cache) maintains a higher sustained throughput at maximum load, demonstrating successful load distribution through replica architecture. Alohomora (no replica) achieves higher peak throughput (111 users/s at 800 users), but shows instability at extreme loads, dropping to 95 users/s at 1000 users. Replica coordination introduces approximately 300 ms of additional latency during steady-state operation. Although replica architecture adds latency overhead, it provides better stability at extreme loads, avoiding the performance degradation seen in the no-replica variant.

Latency Predictability Analysis. The P95 latency analysis (Figure 6) reveals critical system stability differences. Alohomora Standard exhibits extreme variability with a severe spike to 4000 ms at 300 users, followed by recovery and relatively stable performance (900–2000 ms) across higher loads. This suggests an initial replica synchronization stress that resolves as the system stabilizes. Alohomora (without cache) maintains consistently high P95 latency (2700–3300 ms) across all loads, demonstrating the latency cost of direct main server access. Introspection systems maintain exceptional latency stability at approximately 1000 ms P95 across all load levels, demonstrating the predictability advantage of centralized OAuth processing.

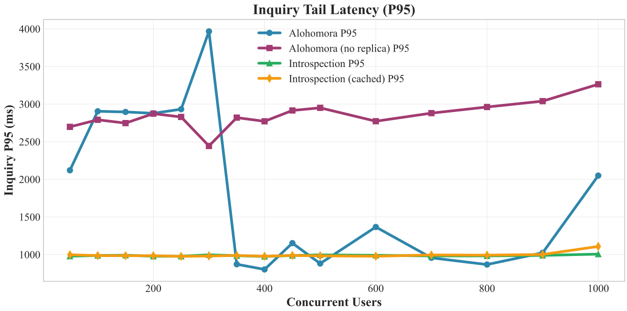

Figure 6. Latency distribution analysis: P95 tail latency times (95% of requests completed in the given time).

Caching Effectiveness Analysis. The minimal performance difference between cache-based and non-cache introspection variants reveals that under high concurrent load, cache contention, and synchronization overhead appear to offset cache hit benefits. The employee onboarding workflow's large token space may result in low cache hit rates, reducing cache effectiveness. The 10-s TTL may be suboptimal for this workflow pattern, suggesting the need for adaptive caching strategies.

5.5. Performance Analysis with Geographic Dispersion

A distributed testbed across Amazon Web Services Elastic Compute Cloud (AWS EC2) assesses Alohomora's performance under realistic network conditions. The testbed spans three geographically dispersed regions. The deployment architecture consists of four primary components, each running on a dedicated t2.micro instance with one virtual central processing unit (vCPU) and one gigabyte of memory, while Alohomora's main server runs on t2.medium, powered by two vCPUs and four gigabytes of memory. The Alohomora main server operates in the US East region (Virginia), serving as the authoritative token validation endpoint, while the replica cache server is deployed in the Middle East South to evaluate geographic load distribution effectiveness. Two application servers representing distinct microservices in the employee onboarding workflow are also positioned in the Middle East South region, creating realistic inter-service communication patterns with network latencies. The user requests are sent from the EU East region to the application servers in the Middle East South region. The application servers mainly communicate with their replica in the same region, with a fallback to the main Alohomora server in the US East region.

All components communicate over HTTPS with TLS 1.3, using nginx as a reverse proxy to provide production-grade request handling and load balancing capabilities. The replica synchronization interval is configured to 60 s in all experiments unless otherwise indicated. Performance measurements are collected using a dedicated load generator that executes from the same EU East region, systematically varying concurrent user loads of users over various test durations using a linear, an exponential, and a step-wise arrival pattern to simulate gradual system scaling.

5.5.1. Request Performance Lifecycle

The request performance analysis starts with simulating client applications sending requests to the first service on a two-step workflow. Each user client executes a complete workflow lifecycle that mirrors two steps of an employee onboarding scenario. The request sequence begins with establishing TLS connections to measure connection setup overhead, including DNS resolution, TCP handshake, and TLS negotiation latencies. Once connected, the user initiates a session creation request to the first application server (App1) in the Middle East South region, providing a unique user identifier. Upon successful session creation, the user registers an authentication token with Alohomora through App1, associating the token with the workflow identifier, function identifier, and system identifier. After token registration, a warmup period allows the replica server to synchronize the token state from the main Alohomora server. The warmup is adjusted to see the effect of this parameter (results in Section 5.5.4). Following the warmup phase, the user establishes a TLS connection to the second application server (App2) and begins executing function calls that require token validation. App2 validates each incoming token by querying the single local replica cache server first, falling back to the main Alohomora server in US East only when the replica cannot resolve the token. This replica-first validation strategy is designed to minimize latency by leveraging geographic proximity and reducing load on the centralized main server.

5.5.2. Collected Metrics

Throughout the workflow execution, comprehensive metrics are collected, including CPU usage, memory consumption, token registration latency, token query latency, cache hit rates, and connection establishment times. As presented in Section 5.5.3, the load generator coordinates multiple concurrent user threads, each following this lifecycle with arrival times determined by the configured traffic pattern (linear or exponential), introducing small random jitters to avoid synchronized burst behavior that could artificially stress the system.

5.5.3. Request Performance Results

Performance evaluation results for linear and exponential arrival patterns are presented in Figures 7–9. Figure 7 shows the box plot for the major operations measured during the experiments, along with the operational latency percentiles.

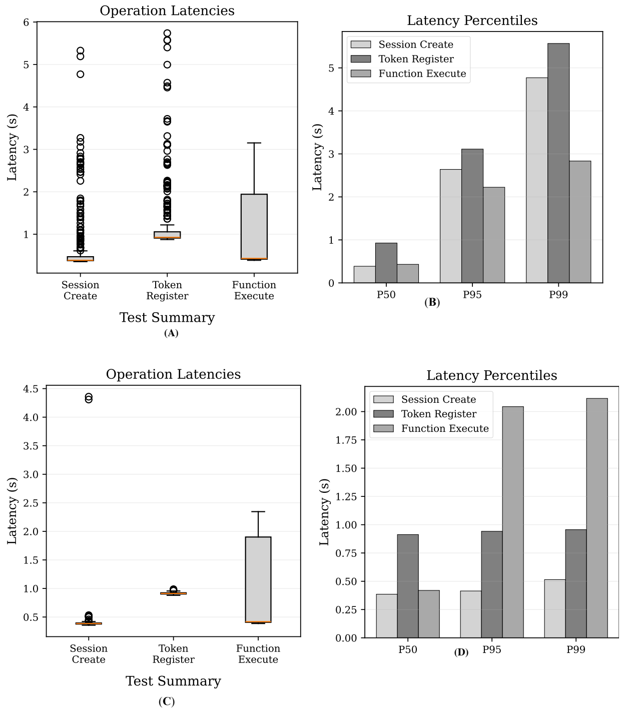

Figure 7. Comprehensive performance metrics under linear and arrival patterns. (**A**) linear operational latencies, (**B**) linear latency percentiles, (**C**) exponential operational latencies, (**D**) exponential latency percentiles.

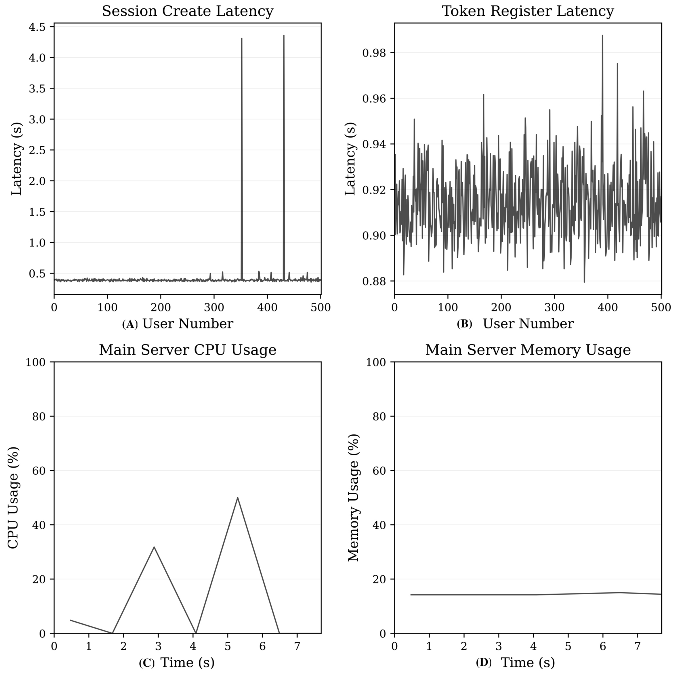

Figure 8. Resource utilization under linear pattern. (**A**) session creation latency, (**B**) token registration latency, (**C**) CPU utilization, (**D**) memory utilization.

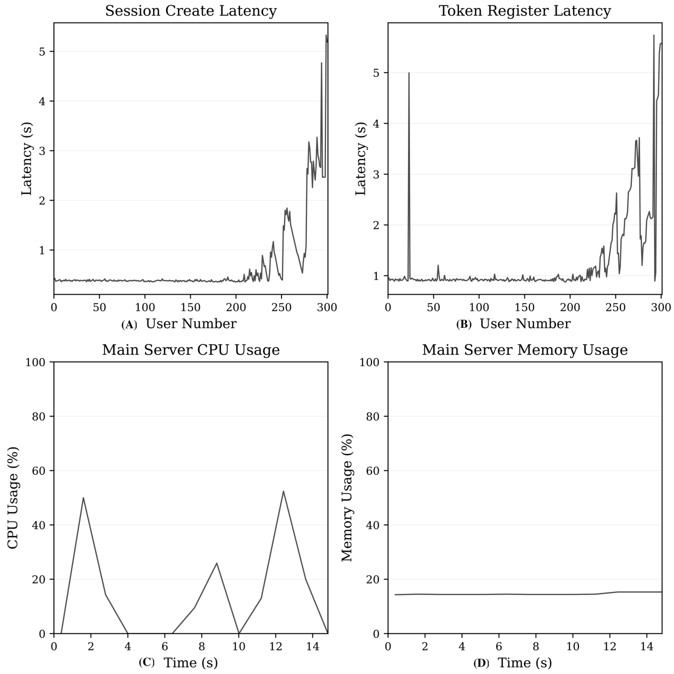

Figure 9. Resource utilization under exponential pattern. (**A**) session creation latency, (**B**) token registration latency, (**C**) CPU utilization, (**D**) memory utilization.

Comparing linear versus exponential arrival patterns reveals distinct performance characteristics. Linear arrival exhibits stable behavior with a mean token registration latency of 914.5 ms, a mean token query latency of 847.3 ms, and a 72.2% cache hit rate. Exponential arrival, simulating burst traffic, degrades performance substantially. Mean token registration latency increases 38% to 1261.7 ms, while session creation P95 latency reaches 2639.2 ms, indicating a 6.4-fold decrease from linear’s 414.3 ms. The cache hit rate drops slightly to 69.0%, indicating that burst traffic temporarily overwhelms replica synchronization. Despite these degradations, CPU and memory metrics confirm the system handles both patterns without resource exhaustion. Notice that the machines used in the experiments have limited resources. The machines did not execute any other user-defined tasks during the experiments.

5.5.4. Cache Performance Results

To systematically evaluate the replica cache effectiveness under realistic geographic distribution, this section presents the results for four experiments targeting different aspects of cache behavior. Together, these experiments isolate the effects of synchronization timing, request rate, concurrency level, and warmup duration on replica cache effectiveness in geographically distributed deployments.

The first experiment examines the impact of replica synchronization intervals by varying the sync period across 15, 30, 60, and 120 s while maintaining a constant request rate (one request every two seconds), revealing how synchronization frequency affects cache freshness and hit rates. The second experiment sweeps across request arrival rates (request speed values from 0.1 to 4.0 requests per second), investigating cache performance under varying traffic intensities to identify optimal operating points and saturation thresholds. The third experiment tests cache-aware concurrency by scaling from 1 to 50 concurrent users with a variable warmup period. The fourth experiment provides a baseline comparison by running identical concurrency tests with a short warmup (below three seconds), producing cache misses under concurrent load to establish the performance penalty of bypassing the replica.

The cache performance results reveal critical trade-offs between synchronization frequency, warmup duration, and system responsiveness. Table 2 demonstrates that shorter synchronization intervals yield substantially higher cache hit rates (98% at 15 s versus 77% at 120 s) and lower mean latency (580.9 ms versus 967.9 ms), confirming that aggressive synchronization improves cache freshness at the cost of increased network overhead. Table 3 shows variable performance across request rates with hit rates ranging from 59% to 96%, suggesting that certain arrival patterns may temporarily desynchronize replica state or trigger cache eviction under load. Tables 4–6 collectively illustrate the critical importance of warmup duration: with full synchronization (60 s warmup), single-user requests achieve perfect cache hits (100%) and low latency (426.8 ms P95), whereas minimal warmup produces zero cache hits for single users and latencies exceeding 2700 ms, demonstrating a 6.4-fold performance penalty when bypassing the replica.

Table 7 reveals consistent connection establishment overhead averaging 228 ms across concurrency levels, with initial DNS resolution requiring 14.8 ms for the first connection but stabilizing to 2–3 ms for subsequent connections, indicating effective DNS caching. These results confirm that replica caching provides substantial performance benefits in geographically distributed deployments when synchronization parameters are properly tuned to balance freshness and overhead.

Table 2. Impact of replica synchronization interval on cache hit rate and token validation latency (constant request rate of 0.5 requests per second over 300 s).

Sync (s)	Hit Rate	Mean (ms)	P95 (ms)	P99 (ms)
15	0.98	580.9	1904.3	2723.1
30	0.95	593.4	2580.1	2880.7
60	0.90	701.9	2621.3	2683.2
120	0.77	967.9	2681.4	2808.0

Table 3. Cache performance under varying request arrival rates (180 s duration, 60 s sync interval, averaged across three repetitions).

λ (Requests/s)	Hit Rate	Mean (ms)	P95 (ms)	P99 (ms)
0.10	0.86	707.2	1751.8	2446.3
0.25	0.89	658.2	2272.2	2598.1
0.50	0.64	1127.9	1735.3	2541.6
0.75	0.88	650.9	2299.5	2407.6
1.00	0.96	518.1	1125.3	2437.8
1.50	0.59	1233.3	2466.0	2737.8
2.00	0.94	556.0	2215.8	2565.4
2.50	0.65	1112.4	1167.4	2209.7
3.00	0.91	608.1	2329.9	2517.0
4.00	0.96	506.5	672.1	2471.2

Table 4. Cache performance with moderate warmup period (35 s warmup, 60 s sync interval, linear arrival pattern).

Users	Hit Rate	P95 (ms)	P99 (ms)
1	0.40	1072.1	1072.1
5	0.60	1072.1	1746.3
10	0.64	1034.1	4389.5
20	0.68	991.1	2232.0
30	0.69	1013.5	2487.4
40	0.69	1017.2	2528.6
50	0.70	964.4	1259.3

Table 5. Cache performance with full replica synchronization (60 s warmup, 60 s sync interval, linear arrival pattern).

Users	Hit Rate	P95 (ms)	P99 (ms)
1	1.00	426.8	426.8
5	0.52	978.9	1055.5
10	0.70	1148.6	2100.3
20	0.74	1041.4	3021.7
30	0.70	1015.9	1872.5
40	0.69	998.2	2125.5
50	0.68	1016.9	2217.5

Table 6. Baseline cache performance with minimal warmup showing cache miss penalty (short warmup, 60 s sync interval, linear arrival pattern).

Users	Hit Rate	P95 (ms)	P99 (ms)
1	0.00	2724.0	2724.0
5	0.20	2859.2	2859.2
10	0.20	2831.7	2831.7
20	0.15	3127.6	3127.6
30	0.07	2964.1	2995.4
40	0.10	3052.1	3353.1
50	0.16	3012.1	3253.5

Table 7. Connection establishment overhead breakdown across geographic regions (averaged across concurrency levels from Table 5).

Users	DNS (ms)	TCP (ms)	TLS (ms)	Total (ms)
1	14.8	106.9	111.5	233.2
5	3.6	108.9	116.2	233.3
10	3.5	110.5	116.6	229.5
20	3.5	109.9	114.4	228.0
30	3.3	110.4	114.8	230.1
40	2.4	110.5	115.2	229.1
50	2.4	110.2	114.6	226.8

6. Conclusions

Alohomora is a distributed system authentication platform that allows single sign-on, is context-sensitive, and produces workflow-based authentication sessions. The protocols and algorithms for implementing Alohomora show its simplicity and effectiveness while maintaining strong security guarantees under minimal security assumptions.

A core component of Alohomora is a distributed cache. Although the main Alohomora functions exist on a single server, the cache synchronization protocol enables a fast, efficient, and secure distribution of the token validation protocol in group-based regional token database replicas. The design is flexible, allows for any number of replicas to exist, and employs fail-safe mechanisms to suspend replica synchronization when suspecting attacks.

In this article, we also presented an extensive experimental evaluation of Alohomora and a comparison with OAuth as the main state-of-the-art method. Our results show polynomial performance penalties under heavy concurrent user load. We also establish the efficacy of the cache versus main-server fallback mechanism, showing that Alohomora maintains high throughput under high load.

The Alohomora prototype implementation, including the main server, replica synchronization protocols, and experimental evaluation framework, is publicly available as open source at https://github.com/kussl/Alohomora (access on 30 October 2025). This enables researchers and practitioners to reproduce the experimental results, extend the system with additional features, and adapt the architecture for production deployments in their own workflow orchestration environments.

Funding: This work was supported and funded by Kuwait University Research Grant No. RS02/23.

Data Availability Statement: This work has not used any data and did not produce data. However, the source code for the Alohomora prototype, including the main server, replica synchronization protocols, and experimental evaluation framework, is openly available at https://github.com/kussl/Alohomora (access on 30 October 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SSO	Single Sign-On
FIM	Federated Identity Management
IdP	Identity Provider
SP	Service Provider
SAML	Security Assertion Markup Language
XML	Extensible Markup Language
OAuth	Open Authorization

API	Application Programming Interface
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
MAC	Message Authentication Code
AES	Advanced Encryption Standard

References

1. Hardt, D. *The OAuth 2.0 Authorization Framework*; Internet Engineering Task Force (IETF) RFC: Wilmington, DE, USA, 2012; Volume 6749.
2. Recordon, D.; Reed, D. OpenID 2.0: A platform for user-centric identity management. In Proceedings of the Second ACM Workshop on Digital Identity Management, Alexandria, VA, USA, 3 November 2006; pp. 11–16.
3. Groß, T. Security analysis of the SAML single sign-on browser/artifact profile. In Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, USA, 8–12 December 2003; pp. 298–307.
4. Hu, V.C.; Kuhn, D.R.; Ferraiolo, D.F.; Voas, J. Attribute-based access control. *Computer* **2015**, *48*, 85–88. [CrossRef]
5. Rowling, J.K. *Harry Potter and the Philosopher's Stone*; Bloomsbury Publishing: London, UK , 2015; Volume 1.
6. Fett, D.; Küsters, R.; Schmitz, G. The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines. In Proceedings of the 2017 IEEE 30th Computer Security Foundations Symposium (CSF), Santa Barbara, CA, USA, 21–25 August 2017; pp. 189–202. [CrossRef]
7. Fett, D.; Küsters, R.; Schmitz, G. A Comprehensive Formal Security Analysis of OAuth 2.0. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 24–28 October 2016; CCS '16; pp. 1204–1215. [CrossRef]
8. Philippaerts, P.; Preuveneers, D.; Joosen, W. OAuch: Exploring security compliance in the OAuth 2.0 ecosystem. In Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses, Limassol, Cyprus, 26–28 October 2022; pp. 460–481.
9. Hosseyni, P.; Küsters, R.; Würtele, T. Formal Security Analysis of the OpenID FAPI 2.0 Family of Protocols: Accompanying a Standardization Process. *ACM Trans. Priv. Secur.* **2024**, *28*, 1–36 . [CrossRef]
10. Neuman, C.; Yu, T.; Hartman, S.; Raeburn, K. The Kerberos Network Authentication Service (V5). Request for Comments (RFC 4120); RFC Editor, July 2005. Available online: https://www.rfc-editor.org/rfc/rfc4120.html (accessed on 30 October 2025).
11. Fielding, R.; Kaiser, G. The Apache HTTP Server Project. *IEEE Internet Comput.* **1997**, *1*, 88–90. [CrossRef]
12. Migeon, J.Y. *The MIT Kerberos Administrators How-to Guide*; Kerveros Constortium: Cambridge, MA, USA , 2008; Volume 6.
13. Mainka, C.; Mladenov, V.; Schwenk, J.; Wich, T. SoK: Single Sign-On Security— An Evaluation of OpenID Connect. In Proceedings of the 2017 IEEE European Symposium on Security and Privacy (Euro S&P), Paris, France, 26–28 April 2017; pp. 251–266. [CrossRef]
14. Fett, D.; Kusters, R.; Schmitz, G. SPRESSO: A Secure, Privacy-Respecting Single Sign-On System for the Web. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 12–16 October 2015; CCS '15; pp. 1358–1369. [CrossRef]
15. Yang, F.; Manoharan, S. A security analysis of the OAuth protocol. In Proceedings of the 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, Canada, 27–29 August 2013; pp. 271–276. [CrossRef]
16. Habiba, U.; Masood, R.; Shibli, M.A.; Niazi, M.A. Cloud identity management security issues & solutions: A taxonomy. *Complex Adapt. Syst. Model.* **2014**, *2*, 1–37. [CrossRef]
17. Hanchate, P. Role Based Provisioning from Oracle Fusion Application to IDCS. 2021. Available online: https://blogs.oracle.com/cloud-infrastructure/post/role-based-provisioning-from-oracle-fusion-application-to-idcs-v2 (accessed on 30 October 2025).
18. Armando, A.; Carbone, R.; Compagna, L.; Cuellar, J.; Tobarra, L. Formal analysis of SAML 2.0 web browser single sign-on: Breaking the SAML-based single sign-on for Google apps. In Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE '08), Alexandria, VA , USA, 27 October 2008; pp. 1–10. [CrossRef]
19. Wang, R.; Chen, S.; Wang, X. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–23 May 2012; pp. 365–379. [CrossRef]
20. Lux, Z.A.; Thatmann, D.; Zickau, S.; Beierle, F. Distributed-Ledger-based Authentication with Decentralized Identifiers and Verifiable Credentials. In Proceedings of the 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 28–30 September 2020; pp. 71–78. [CrossRef]

21. Patel, S.; Sahoo, A.; Mohanta, B.K.; Panda, S.S.; Jena, D. DAuth: A decentralized web authentication system using Ethereum based blockchain. In Proceedings of the 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 30–31 March 2019; pp. 1–5.
22. Sunyaev, A. Distributed ledger technology. In *Internet Computing*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 265–299.
23. Chowdhury, M.J.M.; Ferdous, M.S.; Biswas, K.; Chowdhury, N.; Kayes, A.; Alazab, M.; Watters, P. A comparative analysis of distributed ledger technology platforms. *IEEE Access* **2019**, *7*, 167930–167943. [CrossRef]
24. Bhargava, B.; Zhong, Y. Authorization Based on Evidence and Trust*. In *Data Warehousing and Knowledge Discovery*; Kambayashi, Y., Winiwarter, W., Arikawa, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2002; pp. 94–103.
25. Mohammad, A. Distributed authentication and authorization models in cloud computing systems: A literature review. *J. Cybersecur. Priv.* **2022**, *2*, 107–123. [CrossRef]
26. Li, A.S.; Safavi-Naini, R.; Fong, P.W.L. A Capability-based Distributed Authorization System to Enforce Context-aware Permission Sequences. In Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies (SACMAT ’22), New York, NY, USA, 8–10 June 2022; pp. 195–206. [CrossRef]
27. de Almeida, M.G.; Canedo, E.D. Authentication and authorization in microservices architecture: A systematic literature review. *Appl. Sci.* **2022**, *12*, 3023. [CrossRef]
28. Garfinkel, T.; Pfaff, B.; Chow, J.; Rosenblum, M.; Boneh, D. Terra: A virtual machine-based platform for trusted computing. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 19–22 October 2003; pp. 193–206.
29. Zeldovich, N.; Kannan, H.; Dalton, M.; Kozyrakis, C. Hardware enforcement of application security policies using tagged memory. In Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI’08), San Diego, CA, USA, 8–10 December 2008; pp. 225–240.
30. Almohri, H.M.J.; Watson, L.T.; Evans, D. Misery Digraphs: Delaying Intrusion Attacks in Obscure Clouds. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1361–1375. [CrossRef]
31. Almohri, H.; Watson, L.; Evans, D.; Billups, S. Dynamic System Diversification for Securing Cloud-based IoT Subnetworks. *ACM Trans. Auton. Adapt. Syst.* **2022**, *17*, 1–23. [CrossRef]
32. Sirbu, M.A.; Chuang, J.I. Distributed authentication in Kerberos using public key cryptography. In Proceedings of the SNDSS’97: Internet Society 1997 Symposium on Network and Distributed System Security, San Diego, CA, USA, 10–11 February 1997; pp. 134–141.
33. Bernstein, D.J. The Poly1305-AES message-authentication code. In Proceedings of the 12th International Conference on Fast Software Encryption (FSE’05), Paris, France, 21–23 February 2005; pp. 32–49.
34. Garrison, W.C.; Shull, A.; Myers, S.; Lee, A.J. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 819–838.
35. Tridgell, A.; Mackerras, P. *TR-CS-96-05. The Rsync Algorithm*; Joint Computer Science Technical Report Series; The Austrailian National University: Canberra, Australia, 1996; pp. 1–6.
36. Kwak, B.J.; Song, N.O.; Miller, L.E. Performance analysis of exponential backoff. *IEEE/ACM Trans. Netw.* **2005**, *13*, 343–355. [CrossRef]
37. Cooper, D.; Santesson, S.; Farrell, S.; Boeyen, S.; Housley, R.; Polk, W. RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008. Available online: https://www.rfc-editor.org/rfc/rfc5280.html (accessed on 30 October 2025).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.