# Distributed Session Management

Mance Harmon & David Waite, Ping Identity

## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>2</td>
</tr>
<tr>
<td>Background</td>
<td>2</td>
</tr>
<tr>
<td>Single Logout Today</td>
<td>3</td>
</tr>
<tr>
<td>Front Channel Solutions</td>
<td>3</td>
</tr>
<tr>
<td>Back Channel Solutions</td>
<td>4</td>
</tr>
<tr>
<td>Distributed Session Management</td>
<td>4</td>
</tr>
<tr>
<td>Designing a Better Solution</td>
<td>4</td>
</tr>
<tr>
<td>A New Approach: The Broad Strokes</td>
<td>5</td>
</tr>
<tr>
<td>Leveraging Distributed Consensus</td>
<td>7</td>
</tr>
<tr>
<td>Conclusion</td>
<td>10</td>
</tr>
</tbody>
</table>
Abstract

Single logout (SLO) enables users and administrators to actively end a user's logged-in session across multiple applications. SLO is a necessary component of a Continuous Authentication solution that replaces the use of time-bound sessions with sessions that are terminated by exception.

Single logout mechanisms differ by both client type (web browser or native application) and protocol (SAML, OAuth, OIDC). Some of the mechanisms are brittle and can leave a user or an admin wondering if the mechanism worked.

This paper discusses a distributed session management (DSM) solution, leveraging a next-generation hashgraph distributed consensus algorithm that enables fine-grained session management (e.g., logout, extension, suspension). This solution works across different protocols and client types, and provides administrators certainty (a cryptographic timestamp) that session commands are delivered to the applications. Further, DSM facilitates Continuous Authentication not only by enabling reliable SLO, but by also providing a communications channel that enables the sharing of risk signals between clients, applications and identity providers.

Background

Generically, the term session refers to a period of time in which information is exchanged between two or more communicating participants. As it relates to identity in computing environments, a session refers to the period of time that a user agent (such as a web browser) is allowed to act on behalf of a user for accessing a resource. For example, when a user attempts to access an application, the application first asks for an identifier, such as a username or email address, then requires the user to authenticate his identity. If successful, the application establishes a session, during which time the user is granted access to the application. For our purposes, we define this as an application-scoped session or simply application session.

An application-scoped session persists until either the user manually ends the session by logging out, or the application ends the session for reasons dictated by policy (e.g., user inactivity). Typically, application sessions have a predefined duration. Administrators establish policy that governs the duration of application sessions, what might end a session prematurely, and what is required to re-establish a session upon termination.

There may be many reasons why administrators may want to actively end an application session. For example, consider the case of Alice, an enterprise employee using a web application. An application administrator might choose to end an active application session if Alice’s employment is terminated, thereby removing access. Or an administrator might end an application session if they learn that the IP address associated with Alice’s web browser is used by attackers.

In traditional client–server applications, the mechanisms for managing application sessions are proprietary to each application, if such mechanisms exist at all. However, web applications use a common web browser to render the user interface for each application. As a result, web applications use common mechanisms for session management. For example, an application might set a session cookie in browser memory. When a session is ended, either by the user or by the application, the cookie is deleted.

The need for tools to help manage user access grew with the rise in the number of web applications. Web access management (WAM) emerged as a technology category, and for a time, it optionally provided administrators a way to manage sessions for web applications hosted inside the enterprise.

In the early 2000s, the technology landscape changed in two fundamental ways. First, companies began to use applications hosted on the Internet by third parties. The technical solutions used by WAM products for session management no longer worked for software-as-a-service (SaaS) applications hosted in different security domains. Second, the proliferation of mobile devices resulted in both SaaS and web applications using native mobile applications as the user interface, rather than a common web browser.

The shift to SaaS and mobile native applications has resulted in a set of identity protocol specifications, each of which specifies different single logout mechanisms that are both brittle and complicated to implement. As a result, there is a need for a session management mechanism that has the simplicity of the earlier WAM-based systems, but also works across security domains, across client types (i.e., browsers and native apps), and across identity protocols.
Single Logout Today

Federated single sign-on (SSO) is the industry-accepted way for identity to be shared, whether between partners, to externally hosted services, or even to integrate internal applications.

The various standard protocols for SSO work by exchanging tokens, which serve as a secure way to share user authentication information and identifying attributes between an identity provider (IdP) and an application.

When using single sign-on technology, a user authenticates with the IdP to create an authentication session. An authentication session is the period of time during which the IdP provides identity assertions (tokens) to the applications that rely on the IdP. These are referred to as the relying parties (RPs). When the user attempts to access an application the first time without an appropriate application-scoped session, they are redirected to the IdP for authentication. If the user has already authenticated (and thus has an authentication session) then they are seamlessly redirected back to the application with an identity assertion that represents the user's identity. The application validates the identity assertion and logs the user into the application by creating an application session. If the user accesses three applications, then a total of four sessions may exist: the authentication session, and an application-scoped session for each application.

If a user or administrator wishes to enable single logout and thus end all sessions simultaneously, then some form of communication is required with the IdP and each application. Unfortunately, simply revoking the authentication session does nothing to revoke each of the application sessions. Each application must be notified that its application session should be terminated. If the user wishes to end the session, the user can manually click a logout button in the user interface rendered by the web browser. The user is required to do this separately for each application, as well as the IdP. However, there are many cases where the user or administrator wishes to logout of multiple application sessions automatically, without requiring the user to manually click a logout button for each application. Modern identity protocols (e.g., SAML, OAuth, OIDC) specify, to varying degrees, one of two fundamental mechanisms for single logout.

Front Channel Solutions

The web browser may be used as a communication channel to the applications. This is normally referred to as the front channel. Front channel logout requests are primarily user initiated, not initiated by a system administrator. The user navigates to a global logout page hosted by the identity provider. When the global logout button is clicked, the web browser is used to open a specific logout URL for each application. When loaded, the logout URL results in the application ending the application session for the user. In front channel solutions, the application session information is stored in a browser cookie. When logging out of the session, the cookie is deleted from browser memory. (Ideally, the application keeps a list of sessions that have been revoked, then validates all requests against the revocation list.)

The web browser must load the logout URL of every application with an active session. To achieve this, several technical solutions have been proposed.

- **iFrame** – The identity provider causes the web browser to open a separate iFrame for each application. Each iFrame loads the logout URL for its application.

- **Form Post/Redirection** – When the browser loads the logout URL for an application, the browser also passes the logout URL of the next application in the list of applications with active sessions. After each logout URL is successfully loaded, the browser is automatically redirected to the next logout URL in the list.

- **Logout Images** – The identity provider dynamically generates a web page with a list of image URLs, each of which corresponds to an application logout URL.

Front channel solutions are brittle, do not guarantee effectiveness, and require that the user be present at the browser. For example, the user might close the browser before logouts are completed. And logout URLs might not successfully load for other reasons. In either case, the status of application sessions will be unknown to the user and administrator.
Back Channel Solutions

Many times, there is a business requirement for verifiable single logout. In these cases, a direct API call is made from the identity provider to the application. This is referred to as the back channel. For example, when an employee is terminated, a system administrator might need to immediately end an application session in an application storing sensitive information. Or an identity provider might determine that the risk of fraud has exceeded some threshold, resulting in the need to end all application sessions for one or more users. In these cases, the identity provider sends a message directly to a logout API hosted by each application.

In front channel solutions, the specific application session to terminate is found by referencing the application session cookie stored in the user's browser. When a single logout is requested, the application uses the cookie to know exactly which session to terminate. However, when an identity provider makes a back-channel call to the application, the application is burdened with maintaining a list of session identifiers for which sessions should be revoked. User identifiers associated with each resource request are compared to the user identifiers in the revocation list. If there is a match, then the application session is ended.

Distributed Session Management

Single Logout is the simplest feature of session management. The proposed approaches for Single Logout are either brittle or cumbersome for application developers and, therefore, inconsistently deployed.

However, if the complexities of implementing distributed session management were mitigated, then additional features would provide a benefit. Consider applications concurrently relying on a single IdP and its associated authentication session. Each application sets an inactivity timer for its application session.

For example, assume a user logs into Salesforce.com. An application session is created with an inactivity timer of 15 minutes. If the user doesn’t interact with the Salesforce application after 15 minutes, then the application session is automatically terminated. This is a security mechanism intended to prevent attackers from exploiting an open browser on an unattended computer. However, in an authentication session, it is often the case that the user hasn’t left the computer, but is simply using a different application.

In these cases, it is of value to maintain an authentication session inactivity timer that can be referenced and updated by applications used during the authentication session. In this way, application sessions are not terminated if the user is simply using a different application.

Designing a Better Solution

Front channel solutions use a common web browser for storing authentication and application session cookies. In other words, they have a simple mechanism for correlating users with sessions. If an IdP redirects the browser to the session logout URL of an application, the application uses the application cookie in browser memory to know which application session to terminate. However, using a web browser as a communications channel lacks verifiability. The IdP doesn’t know when or if the application session logout succeeded. If the browser is closed or communication is interrupted, sessions may not be terminated, and their status will be unknown to the IdP.

Back channel solutions provide verifiability, but lack a simple mechanism for correlating users with sessions. Either the application provides the application session information to the IdP, and the IdP maintains a correlation between the authentication and application sessions, or the application maintains a list of user identifiers reported by the IdP as being revoked.

An ideal solution provides both a mechanism for simply correlating users with application sessions AND verifiability that the application session logout request has been received by the applications, as shown in Figure 1 on the next page.
A New Approach: The Broad Strokes

One approach simplifies the problem by abstracting away the complexities of the communication channels, while concurrently providing shared state information (enabling simple correlation of users with application sessions) and cryptographic verifiability. Specifically, using a distributed consensus algorithm enables each entity to securely share session state information and to know for certain when each application receives an application session logout request.

In a distributed session management solution, each IdP and application maintains a local copy of a replicated session database. (Figure 2 shows a conceptual architecture of distributed session management.) When an IdP creates an authentication session, the session ID (and appropriate metadata) is recorded in the session database. The session record is then automatically replicated to the local session database of each application. During SSO, the IdP then sends a token with the session ID to the application. Upon receipt, the application can introspect the token and use the session ID to lookup the authentication session record in its local session database. Subsequently, based on policy agreed to by the IdP and application administrators, applications can at any time check the status of the authentication session in their local copy of the session database.

---

**Figure 1:** An ideal distributed session management solution provides a mechanism for simply correlating users with application sessions AND verifiability that the application session logout request has been received by the applications.

**Figure 2:** A distributed session management architecture.

---

<table>
<thead>
<tr>
<th></th>
<th>FRONT CHANNEL</th>
<th>BACK CHANNEL</th>
<th>IDEAL SOLUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SHARED STATE</strong></td>
<td>✓ YES</td>
<td>✓ NO</td>
<td>✓ YES</td>
</tr>
<tr>
<td><strong>VERIFIABLE</strong></td>
<td>✓ NO</td>
<td>✓ YES</td>
<td>✓ YES</td>
</tr>
</tbody>
</table>
Session Logout
If an authentication session logout is triggered manually by a user or system administrator, or automatically by policy in response to an elevated threat, the IdP merely changes the status of the authentication session in the local session database, and those changes are replicated to all interested parties. If a user attempts to access an application, the application checks the status of the authentication session in the local session database, then denies access based on the revocation status of the authentication session.

By adding logic at the IdP that correlates user identifiers with active authentication sessions, it becomes practical to perform a single logout for a user across all active authentication and application sessions, and across all client types (web browsers and native mobile apps) for all identity protocols (SAML, OAuth, OIDC). In addition, using the appropriate distributed consensus technology, the IdP knows for certain when each application receives the single logout notification.

Continuous Authentication
A reliable “kill switch” is a necessary component of a Continuous Authentication solution. Continuous Authentication minimizes the need for active authentication on the part of the user. It uses information such as context (e.g., IP address, time of day, location), past behavior patterns and passive biometrics to continually update a Level of Assurance in a user’s identity.

Using Continuous Authentication eliminates the need for time-based sessions. Instead, sessions persist until they are intentionally ended because the Level of Assurance has dropped to an unacceptably low level. When this happens, Distributed Session Management provides a “kill switch” capable of ending a user’s sessions with certainty.

Additionally, DSM provides a powerful mechanism for sharing risk signals useful to a risk engine for calculating the Level of Assurance. SaaS applications might report when user behavior is anomalous, such as accessing a rarely used part of the application or abnormally downloading a batch of records. Similarly, apps and devices might share IP address, geolocation, the results of facial recognition, or other signals that might be useful for ongoing device and user authentication.

Advanced Session Management
With such a powerful session management mechanism, it is possible to implement other features not previously practical.

Session Extension
An activity flag can be attached to an authentication session record. Again, based on policy, if a user is actively using an application, the application can update the activity flag for the authentication session. Applications can then decide to revoke or extend application sessions based on the activity in the authentication session.
Session Suspension
Consider the case of a colleague or child wishing to borrow a mobile device. The primary user might wish to temporarily suspend access to applications, but not fully log out of application sessions altogether. Changing the status of an authentication session to “suspended” would indicate to the applications to deny access, but not to destroy state information contained in the current application session.

Level of Assurance (LOA)
The IdP might include the level of assurance in the user’s identity. If the LOA changes during the life of an authentication session, the changes in the LOA are automatically reflected in the local database of each application. This makes it possible for the applications to dynamically change access privileges.

Fine-grained, Dynamic Attributes
In a way similar to LOA, identity attribute information can be communicated to the applications in more detail. If those attributes change during the course of a session, the application can dynamically change access privileges accordingly.

Leveraging Distributed Consensus
Distributed session management is made practical through the use of new distributed consensus technology. While it is technically possible to implement a distributed session management system using a central server to manage the session database, there are a number of compelling reasons against doing so.

The following are minimum requirements needed to implement distributed session management:

- Low computational requirements (does not require proof of work).
- Reliable broadcast of changes to the session database (i.e., create/update/delete). The communication is considered reliable if it has:
  - Resilience to DoS attacks and machines going offline.
  - No single point of failure.
  - A proof of receipt. Interested parties must receive a cryptographic receipt of changes to the database (i.e., “I know that interested parties have received my changes, and I can prove it.”).
  - A proof of transmission.
- Each event must be timestamped.
  - The community must agree on timestamping. The timestamps must not be influenced by clock drift.
  - The timestamps must be trusted; timing and ordering cannot be chosen by a random party. (Members of a federation may trust each other enough to join the federation, but may not trust each other enough to be legally bound by a timestamp created unilaterally by another member.)
- Scalability - many changes to the database per second and many participants.
- An immutable record of events for auditing purposes.
- Distributed trust.
- Reliable storage of the session database with high availability. By definition, a distributed database is more reliable and more available than a central database.
There are a number of different mechanisms that might be considered for building a distributed session management solution. A discussion of these follows, as well as a comparison chart as shown in Figure 4.

<table>
<thead>
<tr>
<th>Distributed Session Management Requirements</th>
<th>Central Server</th>
<th>Leader-based &amp; non-PoW Blockchain</th>
<th>Proof of Work Blockchain (e.g., Ethereum)</th>
<th>Bitcoin Blockchain</th>
<th>Hashgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low computation</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Resilience to DoS</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>No single point of failure</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Cryptographic proof of receipt</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Cryptographic proof of transmission</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Trusted consensus timestamps</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Scalable (both TPS and members)</td>
<td>☑️ YES</td>
<td>✗ NO (leader)</td>
<td>☑️ YES</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Immutability record for audit</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Distributed trust</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
<tr>
<td>Reliable storage and high availability</td>
<td>☑️ NO</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
<td>☑️ YES</td>
</tr>
</tbody>
</table>

Figure 4: There are a number of different mechanisms that can be considered for building a distributed session management solution.

Note: It may be possible to build some of the missing properties on top of the distributed database system, but they aren’t inherent in the system.

Central Server
A central server may seem like a potential option, but a closer look reveals that it fails to meet several requirements. For starters, a central server is potentially a single point of failure by definition. Failure may happen as a result of accidental crashes or denial of service attacks.

One might argue that relying parties must already depend on the IdP, also a central server, for SSO. However, the impact of downtime of a session management system that provides shared state for all RPs is much greater than the potential impact of downtime for an IdP. Downtime of a central session management server could lead to 100% downtime across all services.

Also, a central session management server does not scale as well as a distributed solution. A central server must service 100% of the relying party requests. In addition, the RP requests are made to a remote server, which increases latency for every communication.
If the record of changes to the session database is centrally managed, then it is not immutable. In this case, the RPs must track communication with the session management server and require signed receipts for every communication to achieve non-repudiation.

Further, relying parties may not wish to legally trust the server operators to create a timestamp for changes to the database. And if the session management server operator is not the IdP, then the IdP must trust the server operator to execute session logout requests.

**Leader-based Systems and Systems without Proof of Work**

Distributed consensus algorithms have been used for decades to ensure fault tolerance and consistency of databases. The Paxos and Raft family of distributed consensus algorithms are the most widely used examples. For example, Paxos is used in Cassandra, Google's Spanner database, and also in their distributed locking service, Chubby.

This generation of algorithms is characterized by a need for a single leader that serves as a central clearinghouse for all transactions. The leader is responsible for the bulk of network communication while elected, and therefore highly susceptible to DoS attacks. As a result, leader-based systems require a higher degree of trust among participants, and the participants must trust the leader, who has the ability to delay or even discard transactions received from the network.

Because some blockchain systems establish trust in the participants prior to allowing their participation, these systems don’t use proof of work. The participants instead take turns publishing blocks of transaction. As a result, each participant serves as a leader, creating the same set of difficulties described above. Finally, leader-based systems don’t directly provide proof of receipt or trusted timestamping.

**Proof of Work Blockchain**

While blockchain meets many of the necessary requirements, it has several shortcomings. If the members of the community are untrusted, then a proof-of-work mechanism is often required to ensure the community comes to consensus. This is both inefficient and can further degrade throughput and waste resources. Also, blockchain doesn’t provide trusted timestamps nor proof of receipt.

**Hashgraph**

When reviewed against the requirements for distributed session management, hashgraph is the only single solution that meets them all. Consider that it does not require proof of work to create trust among untrusted participants. Also, at no time does hashgraph require a leader. As a result, there is no single point of failure, nor is it susceptible to DoS in the same ways that leader-based systems are. And there is no need for members to trust a leader to change the database. Finally, hashgraph inherently provides trusted, consensus timestamps on every database change.
Conclusion

The shift to SaaS and mobile native applications has resulted in different logout mechanisms making single logout solutions both brittle and complicated to implement. As a result, there is a need for a session management mechanism that has the simplicity of the earlier WAM-based systems, but that also works across security domains, across client types (i.e., browsers and native apps), and across identity protocols.

Distributed session management is designed to bring the equivalent of web access management session controls beyond the firewall to SaaS and native applications.

A distributed session management solution promises both a mechanism for simply correlating users with application sessions AND verifiability that the application session logout request has been received by the applications.

A reliable ‘kill switch’ is a necessary component of a Continuous Authentication solution. Distributed session management provides that ‘kill switch’ through verifiable single logout. In addition, DSM supplies a communications channel that simplifies the sharing of risk signals between clients, applications, identity providers and 3rd-party services, further supporting Continuous Auth.

Ping Identity recommends that the ideal distributed session management solution leverage a next-generation hashgraph distributed consensus algorithm and enable fine-grained session management (e.g., logout, extension, suspension, etc). This approach has been shown to work across different protocols and client types, and provides administrators certainty (a cryptographic timestamp) that session commands are delivered to the applications.

About Ping Identity

Ping Identity leads a new era of digital enterprise freedom, ensuring seamless, secure access for every user to all applications across the hyper-connected, open digital enterprise. Protecting over one billion identities worldwide, more than half of the Fortune 100, including Boeing, Cisco, Disney, GE, Kraft Foods, TIAA-CREF and Walgreens trust Ping Identity to solve modern enterprise security challenges created by their use of cloud, mobile, APIs and IoT. Visit pingidentity.com.