ADSS Server v8.3.14 –

Architecture & Deployment Guide

ASCERTIA LTD

DECEMBER 2025

Document Version- 1.0.0

© Ascertia Limited. All rights reserved.

This document contains commercial-in-confidence material. It must not be disclosed to any third party without the written authority of Ascertia Limited.

CONTENTS

1 INTRODUCTION 4
1.1 SCOPE 4
1.2 INTENDED READERSHIP 4
1.3 CONVENTIONS 4
1.4 TECHNICAL SUPPORT 4
2 EXECUTIVE SUMMARY 5
2.1 ADSS SERVER ARCHITECTURE 5
2.2 API INTERFACES 6
2.3 DEPLOYING ADSS SERVER & SIGNINGHUB 7
3 ARCHITECTURE 9
3.1 CLIENT & ADSS SERVER COMMUNICATION 9
3.2 SMTP DEPENDENCY 10
3.3 SYSTEM ADMINISTRATION 10
3.4 ADSS SERVER TOMCAT CONNECTORS 11
3.5 LOCAL SIGNING USING ADSS SERVER 12
3.6 ADSS SERVER GATEWAY DEPLOYMENT MODEL 13
4 COMMUNICATION 15
4.1 CLIENT AUTHENTICATION & IDENTIFICATION 15
4.2 ADMINISTRATION IDENTIFICATION & AUTHENTICATION 15
4.3 ADSS SERVER MODULES 15
5 HIGH AVAILABILITY 19
5.1 DATA 19
5.2 ARCHITECTURE NOTES 19
5.3 SESSION MANAGEMENT 20
6 KEY MANAGEMENT 21
6.1 SAM SERVICE 21
7 LOCAL SIGNING & GO>SIGN DESKTOP 22
7.1 JAVA DEPENDENCY 22
7.2 END-USER EXPERIENCE 22
7.3 GO>SIGN DESKTOP 22
7.4 GO>SIGN DESKTOP PORTS & PROTOCOLS 23
7.5 SECURITY 23
7.6 TRAFFIC FLOW 24
8 SECURITY 26

FIGURES

FIGURE 1 - WINDOWS SERVICE PANEL ADSS SERVER PROCESS OWNER VIEW 5
FIGURE 2 – SIGNINGHUB AND ADSS SERVER COMPONENTS 7
FIGURE 3 - TYPICAL ADSS SERVER DEPLOYMENT SCENARIO 8
FIGURE 4 - ADSS SERVER TYPICAL DEPLOYMENT ARCHITECTURE 9
FIGURE 5 - ADSS SERVER & CLIENT COMMUNICATION 10
FIGURE 6 - ADSS SERVER SMTP DEPENDENCY 10

FIGURE 7 - ADSS SERVER ADMINISTRATION....11
FIGURE 8 - ADSS SERVER TOMCAT CONNECTORS....12
FIGURE 9 – LOCAL SIGNING....13
FIGURE 10 – ADSS SERVER GATEWAY OR PROXY....14
FIGURE 11 – CLIENT MANAGER....15
FIGURE 12 – GO>SIGN LOCAL SIGNING WORKFLOW....23
FIGURE 13 – GO>SIGN DESKTOP & SERVICE PROCESS FLOW....25
FIGURE 14 – GO>SIGN DESKTOP & SERVICE PROCESS FLOW....26

1 Introduction

1.1 Scope

This manual describes the architecture and deployment scenarios for ADSS Server.

1.2 Intended Readership

This manual is intended for ADSS Server administrators responsible for its configuration. It is assumed that the reader has a basic knowledge of digital signatures, certificates and information security.

1.3 Conventions

The following typographical conventions are used in this guide to help locate and identify information:

- Bold text identifies menu names, menu options, items you can click on the screen, file names, folder names, and keyboard keys.
- `Courier New` font identifies code and text that appears on the command line.
- **`Bold Courier New`** identifies commands that you are required to type in.

1.4 Technical Support

If Technical Support is required, Ascertia has a dedicated support team. Ascertia Support can be reached/accessed in the following ways:

Website	https://www.ascertia.com
Email	support@ascertia.com
Knowledge Base	Ascertia Community Portal

In addition to the free support services detailed above, Ascertia provides formal support agreements with all product sales. Please contact sales@ascertia.com for more details.

When sending support queries to Ascertia Support team send ADSS Trust Monitor logs. Use the Ascertia's trace log export utility to collect logs for last two days or from the date the problem arose. It will help the support team to diagnose the issue faster. Follow the instructions on how to run the trace log export utility

2 Executive Summary

This document describes the architecture and deployment scenarios for ADSS Server (Advanced Digital Signature Services Server) and where relevant Go>Sign Desktop for client-side signing using eID or equivalent hardware devices. Peripheral modules are covered such as database and HSM (Hardware Security Module). However, specific configuration of these is outside the scope of the document and the reader should consult the vendor for instructions on how to deploy in a high availability, fault tolerant configuration.

Architecture descriptions cover the deployment options, which include the simple single instance through to high availability fault tolerance set-ups.

Remote Authorised Signing requires the Go>Sign Mobile App (or third-party application that includes the relevant functionality) to allow end users to authorise their signing requests. The Go>Sign Mobile App is available from Apple and Android stores respectfully.

2.1 ADSS Server Architecture

ADSS Server is a Java 17 EE application that runs on Apache Tomcat. ADSS Server provides advanced cryptographic services such as central or local signing, key generation, certificate management, interfaces to internal or external CA (Certification Authority), OCSP (Online Certificate Status Protocol) and TSA (Time Stamp Authority) services. ADSS Server can be deployed on Windows or Linux Servers. The following depicts all ADSS Server components:

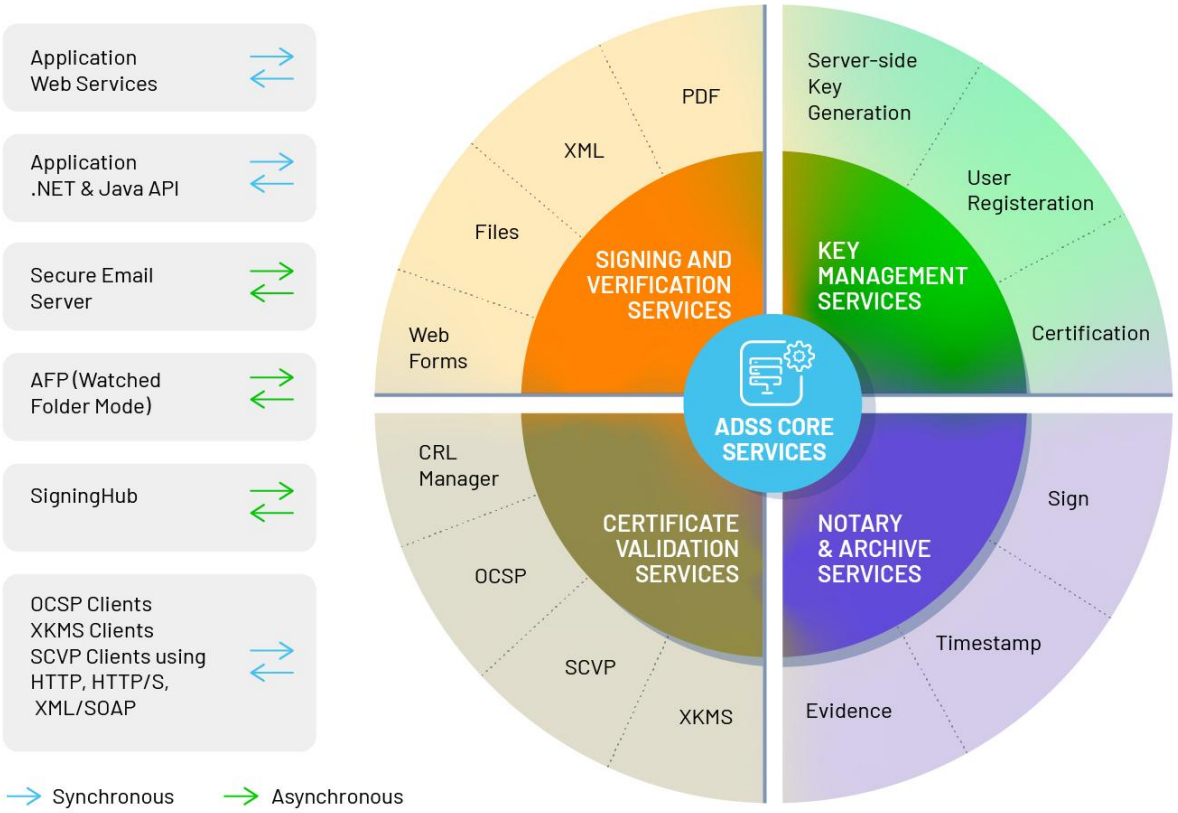

Figure 1 - Windows Service Panel ADSS Server Process Owner View

2.2 API Interfaces

ADSS Server supports both traditional “heavy” web services based upon SOAP, and a lighter REST architectural style API to support Remote Authorised Signing. There is a client SDK that has high-level libraries written in Java and C#. Extensive samples are provided for every service ADSS Server supports.

ADSS Server has a proprietary protocol based upon XML over HTTP(S).

2.2.1 Protocol Support

All APIs are served by the ADSS Server Service module. Underneath it uses standard Tomcat HTTP Connectors to handle the incoming requests for services, whose configuration follows standard Tomcat deployment instructions. The Service Module supports HTTP(S), including mutual TLS.

2.2.2 Remote Authorised Signing REST Architectural Style API

Remote Authorisation Signing (RAS) Service is a purpose-built interface to support remote Qualified signing primarily but will support Advanced level as well. Qualified remote signing requires a specific HSM. One that is Common Criteria certified under EN 419 221-5. This also requires the Common Criteria certified version of Signature Activation Module (SAM) Appliance of ADSS Server. For remote advanced signatures a standard PKCS#11 HSM can be used. The service supports business applications that must manage user registration and lifecycle management, and mobile devices for signing authorisation from end users. The dedicated use case means the API is not intended for generic server-side signing use cases. For this the Signing Service is available to requesting clients.

2.2.3 Signature Activation Module Service

ADSS Server has two varieties of the Signature Activation Module (SAM) for remote authorised signing; Common Criteria certified version (expected certification March 2019) under EN 419 241–2 and non-certified mode. Where certified mode is required then the SAM Appliance must be used. For non-certified mode the SAM can be deployed as a software module alongside other ADSS Server Service modules.

Under normal operation there is never a need for business applications to communicate with the SAM Service. The RAS Service acts as the gateway for this module and handles end user interaction via SMS and SMTP to deliver One Time Passwords during user mobile device registration, and push and receive authorisation requests. However, where required the SAM Service can be used directly by clients. There is a REST architectural style API to support all interfaces.

Note the interfaces are bound by the certified version. Hence, changes to the SAM Service APIs are unlikely.

2.2.4 IETF / RFC Interfaces

Several ADSS Server modules have standard compliant interfaces, removing any dependency on development language. These are OCSP Responder, Time Stamp Authority, Server Certificate Validation Protocol, Simple Certificate Enrolment Protocol, and Certificate Management over CMS (version 1.0).

These interfaces comply with the relevant RFC/IETF standard and have no language dependencies for requesting clients.

2.2.5 W3C XML Digital Signature

ADSS Server supports W3C specification for XML digital signatures and verification thereof. WSDL files are provided for these services.

2.2.6 OASIS DSS/DSS-X

ADSS Server supports OASIS DSS and DSS-X interfaces for signing and verification operations. These have XSD definitions that are packaged with the ADSS Client SDK. Sample high-level code is provided but any requests that conforms to the standard can utilise the services.

2.2.7 Cloud Signature Consortium Protocol

The RAS Service supports both Ascertia proprietary protocols and the Cloud Signature Consortium (CSC) protocol for remote signing. OAuth support for authorisation is not currently available and will be added shortly.

2.3 Deploying ADSS Server & SigningHub

ADSS Server is easy to install and configure. Some initial discussions with Ascertia or its partners may be useful to understand which of the available options best suit the business needs. ADSS Server powers Ascertia SigningHub Enterprise for electronic signature workflow. This diagram shows how SigningHub and ADSS Server work together with various potential business applications. All user and document management are handled inside SigningHub, whilst user key management and cryptographic operations are processed within ADSS Server.

Either a full internal PKI, or one (or more) external PKIs can be used. ADSS Server can either be deployed on the same site or at a different site. This latter option allows ADSS Server to be deployed on premise whilst all the user keys are held within an ISO 27001, SSAE 16, SOC 2 datacentre for example, e.g. Azure.

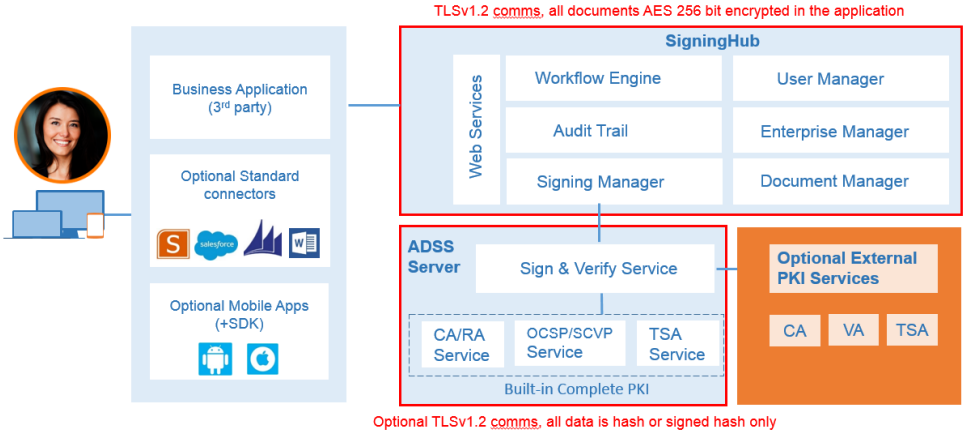

Figure 2 – SigningHub and ADSS Server Components

2.3.1 Resilience & Scalability

ADSS Server is well proven to offer a robust and resilient platform, which supports high availability and fault tolerance. Ascertia's ADSS Server cloud service runs on an Azure instance in East USA and has been running continuously with no unplanned downtime after several years of operation. Other Ascertia partners around the world have similar experiences.

The disc storage and database management system are vital for high availability. Separate ADSS Server instances must be able to access and use the same configuration data to act as an effective, unified system.

Various high availability hardware, network and virtualisation measures can be taken to ensure continued service despite various possible single points of failure.

ADSS Server supports master/slave, active/passive concept for Core and Console modules. Continuous self-monitoring ensures that a failure of the master node results in the automatic promotion of the first slave to takeover.

2.3.2 Typical Deployment

The following diagram depicts a typical deployment of ADSS Server:

this:

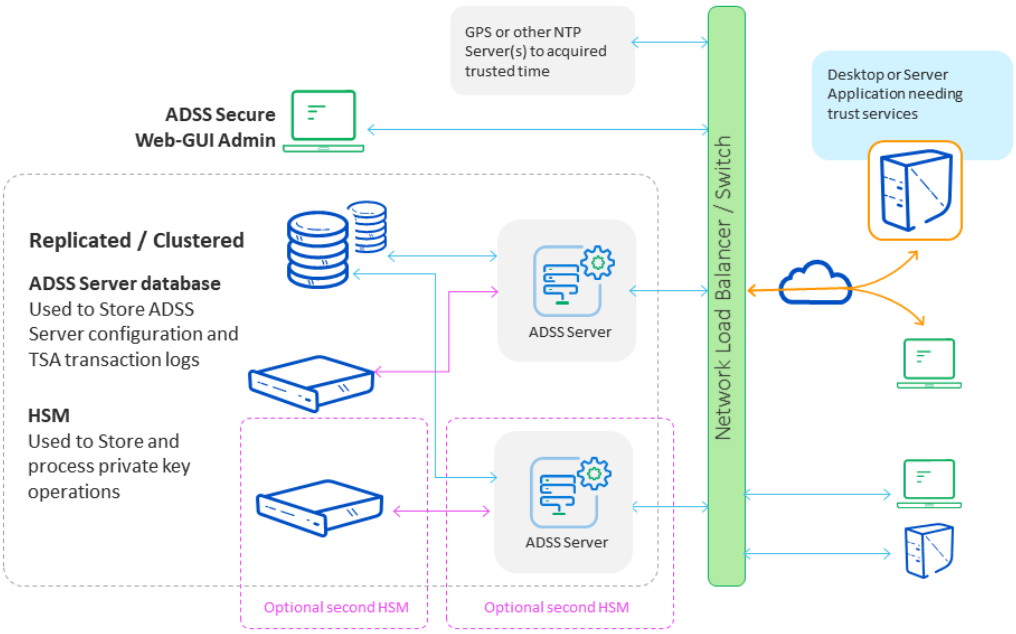

Figure 3 - Typical ADSS Server Deployment Scenario

Administration via the Console is only available using mutual TLS authentication.

Offering an independent DR (Disaster Recovery) facility can raise this availability by removing dependence on a shared database environment.

For a complete list of supported third party elements refer to the ADSS-Server-Installation-Guide.

3 Architecture

For a typical ADSS Server deployment as depicted above in 1.3.2 Typical Deployment, the architecture would match the following:

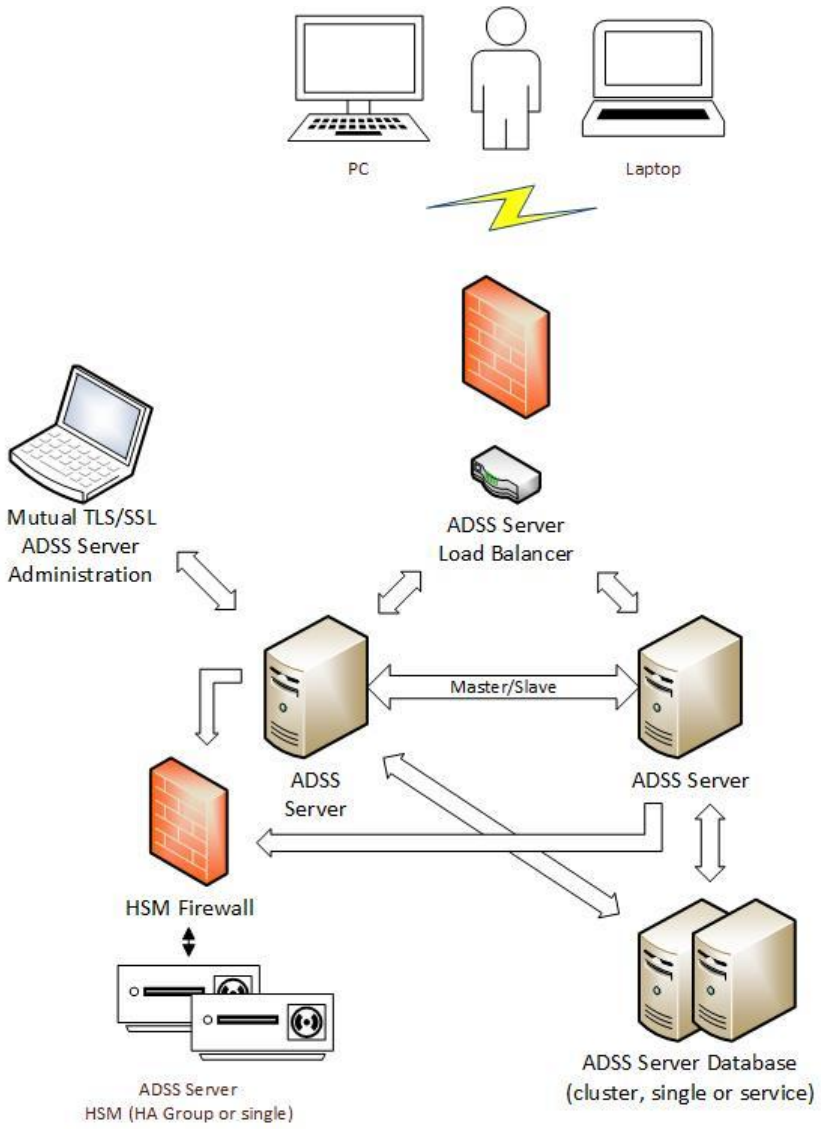

Figure 4 - ADSS Server Typical Deployment Architecture

3.1 Client & ADSS Server Communication

An ADSS Server instigates all communication in this process and requires only standard HTTP(S) over TCP/IP. Note ADSS Server supports client requests on either HTTP or HTTPS protocols. However, it is possible to disable the non-secure channel. In addition, ADSS Server can be configured to accept requests solely on the channel that requires client TLS or server-side TLS, or both.

Regarding bandwidth requirements, it depends on the use case and which ADSS Server modules are enabled. That is, if ADSS Server is required to do the document hashing and formation of the final signed PDF or Word documents the bandwidth requirement is far greater than for OCSP requests.

Bandwidth requirements can vary enormously. For example, clients that use SHA-256 hashing algorithm then the payload of messages sent to ADSS Server are 32-byte blobs for signature. As this is outside the

control of ADSS Server to a point, requirements will vary. The specific services used by clients (in this case SigningHub is the client) of ADSS Server are shown here:

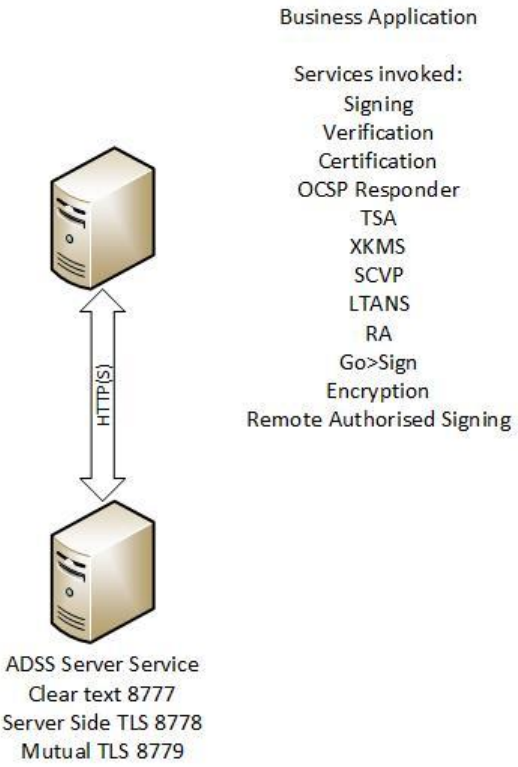

Figure 5 - ADSS Server & Client Communication

3.2 SMTP Dependency

ADSS Server can use SMTP for system alerts to designated administrators. However, this is not mandatory, and SMS is another available option. SNMP is the other alternative/choice:

Figure 6 - ADSS Server SMTP Dependency

3.3 System Administration

ADSS Server administration console is protected by server-side TLS and requires client TLS authentication to access the site. The certificate is mapped to a logical identity in ADSS Server Access Control module once Tomcat has performed the actual authentication.

ADSS Server Console is available directly or via AJP Connector deployed in a DMZ hosted web server. Herewith the architecture for this:

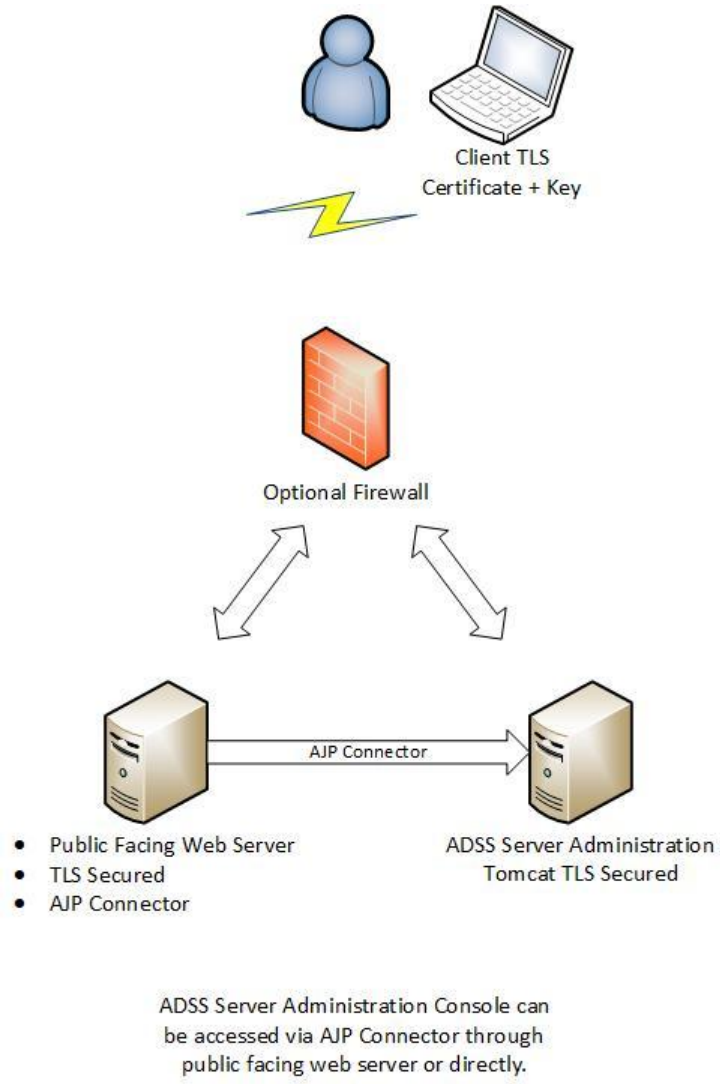

Figure 7 - ADSS Server Administration

Applies to both server-side and local signing operations. Local signing operations rely on ADSS Server Go>Sign Service to perform hash operations, and for server-side client may request ADSS Server to perform the document hashing operation.

3.4 ADSS Server Tomcat Connectors

ADSS Server is a Java 17 EE application hosted by Tomcat application server. In line with best practice, ADSS Server Tomcat uses a combination of HTTP and AJP connectors. AJP Connectors are mandatory if local signing is required in ADSS Server. Here are the connectors and communication flows:

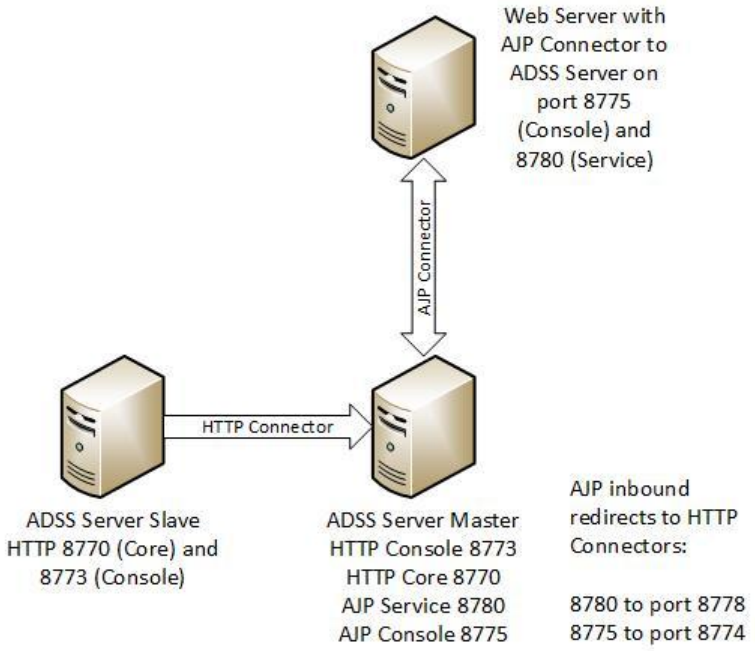

Figure 8 - ADSS Server Tomcat Connectors

3.5 Local Signing using ADSS Server

For specific use cases of local signing, using locally held signature creation devices such as USB Tokens or smart cards, ADSS Server relies on Go>Sign Desktop and Service respectfully. The interaction is covered in more detail in section 6 Local Signing & Go>Sign Desktop. Go>Sign Service has a Viewer module to assist in the display and interaction with the end user. This can be embedded into a business application. Note this is how the SharePoint App works with ADSS Server. For introduction, this is how the combination of ADSS Server, ADSS Server, and Go>Sign components communicate to support the use case:

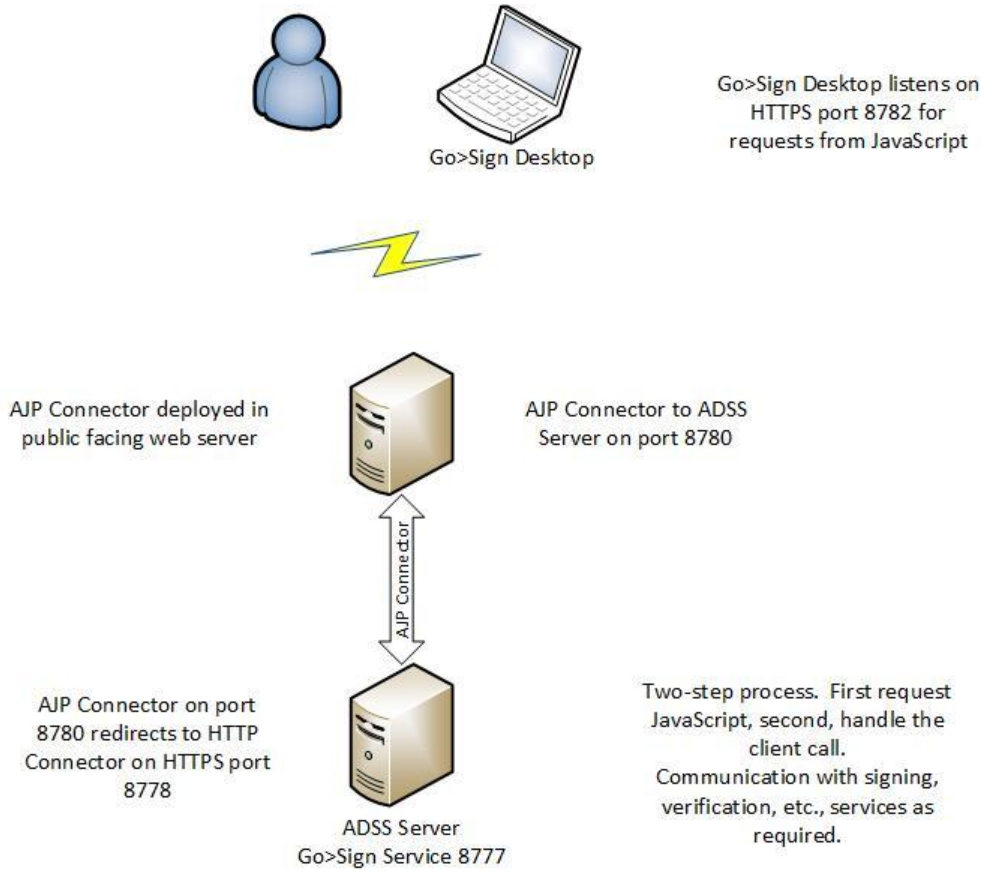

Figure 9 – Local Signing

3.6 ADSS Server Gateway Deployment Model

ADSS Server can operate in a "gateway" mode for the major services. This allows deployment of ADSS Server in different architecture zones, and hence split functionality. For example, OCSP Gateway and OCSP Responder is a specific case where two ADSS Server instances should be deployed and function in different capacities.

The same split also applies to protect access to the HSM. With reference again to OCSP, an OCSP Responder requires access to the HSM for signing the responses. In order to avoid possible exposure of the HSM to the Internet, ADSS Server can operate in OCSP Gateway mode. This allows the public facing ADSS Server instance to accept client's requests, inspect them to check they conform to the required IETF standard, and the act accordingly: reject if the request does not comply to the standard, or pass onto the "backend" ADSS OCSP Server for actual processing and response. The public facing OCSP Gateway then returns the response to the calling client.

ADSS Server can operate in this split architecture for Signing, CSP, TSA, OCSP, and Certification Services. RA to CA split is slightly different because these are different Service modules. Note a RAS Gateway Service is currently under development. The following shows the architecture for these services:

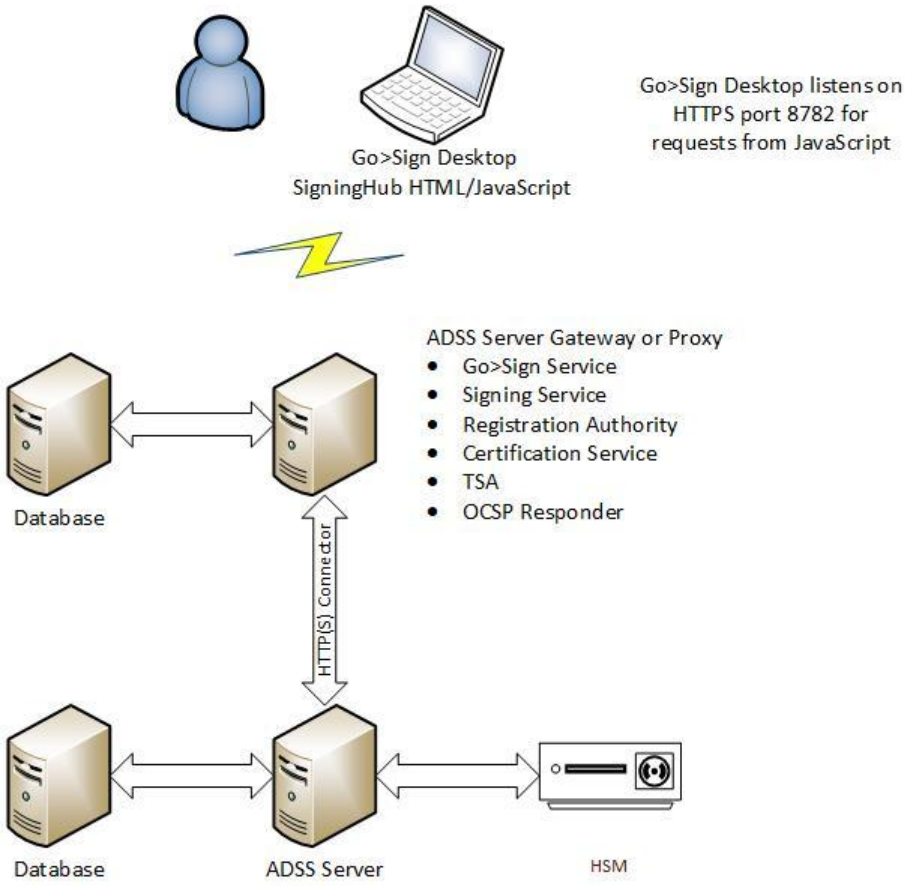

Figure 10 – ADSS Server Gateway or Proxy

A separate database is recommended for this set-up to ensure complete division between the information held, available and used by the two ADSS Server instances. In this architecture the ADSS Server Gateway does not know about the backend ADSS Server and vice versa. This allows separation of tiers for the architecture.

The various communications operate as follows:

- Remote Authorisation Signing Service to Signature Activation Module
- Go>Sign Service to Verification and Signing Services.
- Signing Service to:
  - Remote Signing Service (ADSS Server Signing Service).
  - Remote Authorised Signing Service
- Registration Authority to Certification Service.
- Certification Service to Certification Service.
- TSA Service to TSA Service.
- OCSP Responder Gateway to OCSP Responder Service.
- CSP Gateway to CSP Service.

It is important to note that the ADSS Server Gateway or Proxy will inspect all client requests and handle accordingly. Any calls that do not comply to Ascertia or IETF standards will be rejected. Where possible, for example, OCSP Responder, a well-formed, IETF compliant response will be sent in the case where the request does not comply with IETF standard.

Cases such as Registration Authority, the request must match the Ascertia based protocol and any requests received that do not match this will receive an Ascertia specific rejection.

Only valid requests will be proxied to the backend ADSS Server for processing.

4 Communication

This section details the protocols, ports, and flows of communication that occur within an ADSS Server deployment.

4.1 Client Authentication & Identification

ADSS Server always authenticates a client who attempts to access a service or administration console. Examples of clients are SigningHub, Auto File Processor, and Secure Email Server. Clients are identified via an Originator or Client ID. This alphanumeric string is created by ADSS Server administrator via Client Manager menu:

Figure 11 – Client Manager

All ADSS Server services are exposed via HTTP(S). The receiving host is Tomcat and not proprietary ADSS Server.

Tomcat is also responsible for the security aspect in terms of confidentiality (TLS/SSL) and for ADSS Server Administration Console, mutual TLS/SSL implementation. Once Tomcat has successfully authenticated the client, ADSS Server uses the full certificate to match against a known, and active, client, and thus allow access.

4.2 Administration Identification & Authentication

ADSS Server administrators must be configured via the Access Control menu as seen above. This is where the administrator details are stored, and their respective certificate used in the client TLS authentication process stored.

4.3 ADSS Server Modules

ADSS Server is a Java 17 EE application deployed in three separate Tomcat instances. It consists of many modules that ADSS Server relies on, but which the user never sees. For example, the actual signing service. With respect to the communication ADSS Server dependencies can be broken down into the following categories:

- ADSS Server Console: Web based administration console for system management and configuration.
- ADSS Server Core: Management module that offers services to support all other modules. For example, license management and renewal of system certificates.
- ADSS Server Service: Client facing PKI services such as signing, verification, time stamp authority, and OCSP Responder.

4.3.1 ADSS Server External Interfaces

ADSS Server, where relevant, supports and adheres to IETF and OASIS standards. OCSP, SCVP, and TSA (RFC 6960, 5055, 3161, and 5816 respectively) although accessed using the defined message request syntax of the standard, still use the defined ADSS Server Service ports detailed below. The same applies to OASIS signature and verification requests.

TSA, OCSP, SCVP, and XKMS services can handle multiple profiles, each of which defines a different type of service/configuration. For example, one ADSS Server instance can host multiple time stamp authorities, each with its own respective private signing key, policy, and potentially access control policy.

See here for further details:

ADSS Server Architecture & Interfaces (ascertia.com)

4.3.2 ADSS Server External Dependencies

4.3.2.1 Database

ADSS Server must have access to a suitable database. It maintains and manages its own database connection pool and interaction is based upon JDBC (using Hibernate for database persistence and C3P0 for connection pool management).

Alongside the main ADSS Server database the set-up may be configured to use the full certificate status checking database. This is a separate database and ADSS Server has read only permissions for access.

For SQL Server and Azure SQL Database Windows Authentication is a supported login method alongside local accounts.

ADSS Server Gateway or Proxy can use its own dedicated database if required (recommended).

4.3.2.2 SMTP

If needed for alert notifications to designated Console Operators.

4.3.2.3 SMS Provider

If needed for alert notifications to designated administrators. Clickatell and Twilio are the only implemented service providers currently.

4.3.2.4 SNMP Service

If needed for alert notifications to designated administrators.

4.3.2.5 Publishing Points

Any third-party publishing location. There are several possibilities. For example, Certification Service may be configured to publish issued certificates to an external LDAP directory.

4.3.2.6 NTP Service

Used for a time source for local TSA services. ADSS Server, when configured to use NTP, allows for notifications and ultimately stopping of services, should the local system clock deviate from the time source outside of the stated boundaries.

The TSA service can use the NTP time source directly, but this is not recommended because of performance and throughput.

4.3.2.7 Revocation Information

CRL or OCSP points if required to retrieve certificate revocation information from external CA.

4.3.2.8 External CA

If using an external CA for certificate issuance. Current supported CAs: Microsoft AD CS, PrimeKey EJBCA, GlobalSign EPKI and HVCA, QuoVadis, Entrust Authority Security Manager, Symantec MPKI, DigiCert PKI, SPOC Server and any other ADSS Server CA.

4.3.2.9 External TSA

If using an external TSA for time stamp operations.

4.3.2.10 Hardware Security Module

If using an HSM for secure key generation, use and storage. The interface is based upon PKCS#11 standard for direct communication for most HSM modules, RESTful API for Microsoft Azure Key Vault, and client specific Cavium driver for AWS CloudHSM. MSCAPI/NCG is another option.

ADSS Server uses IAIK Security Provider as a generic interface to PKCS#11 compliant HSMs.

AWS CloudHSM uses the Cavium Java interfaces and is supported on Linux only.

Common Criteria certified Utimaco CP5 CryptoServer SE series HSM use a proprietary Utimaco API, which is an extension of standard PKCS#11. Only Linux deployments of ADSS Server are currently supported with this type of HSM.

4.3.3 ADSS Server Ports & Protocols

Port[1]	Protocol	Initiator	Target	Description
8773	HTTP over TCP/IP	ADSS Server Console – slave instance, (AJP to HTTP Connector).	ADSS Server Console – Master instance.	Used in High Availability deployment for Master Slave communication.
8774	HTTPS over TCP/IP	Administrator web browser	ADSS Server Console	Mutual TLS/SSL to access ADSS Server Console.
8775	AJP Connector over TCP/IP	ADSS Server AJP Connector	ADSS Server Console	Access to ADSS Server Console via AJP Connector deployed on ADSS Server IIS host. Redirects to port 8774.
8770	HTTP over TCP/IP	ADSS Server Core – Slave instance, (AJP to HTTP Connector).	ADSS Server Core – Master instance	Used in High Availability deployment for Master Slave communication.
8771	AJP Connector over TCP/IP	N/A	ADSS Server Core	Legacy – not required for new deployments.

[1] All ports are default and configurable during deployment.

8777	HTTP over TCP/IP	ADSS Server Client. For example, ADSS Server or AFP. ADSS Server JavaScript used during local document view and signature operations.	ADSS Server Service	Non-secure access to ADSS Server Service modules. For example, signing service.
8778	HTTPS over TCP/IP	ADSS Server Client. For example, ADSS Server or AFP. ADSS Server JavaScript used during local document view and signature operations.	ADSS Server Service	Server-side TLS/SSL secured access to ADSS Server Service modules. For example, signing service.
8779	HTTPS over TCP/IP	ADSS Server Client. For example, ADSS Server or AFP.	ADSS Server Service	Mutual TLS/SSL secured access to ADSS Server Service modules. For example, signing service.
8780	AJP Connector over TCP/IP	ADSS Server AJP Connector	ADSS Server Service	Redirect to HTTPS connector running on port 8778 for access to ADSS Server Service modules such as Signing and Verification.

5 High Availability

Production deployments of ADSS Server do not differ from any system in that there must be no single point of failure, and resilience should form part of the solution. Fault tolerance does not necessarily translate into 100% uptime, and even with the best intentions and set-up there remains a possibility that things can go wrong. Therefore, as with any deployment Ascertia recommends that for the highest form of fault tolerance and availability at least two entirely separate data centres are used to host the solution.

5.1 Data

All data (configuration and user) that ADSS Server uses and needs, is stored in the respective databases. This ensures there is no reliance of configuration files in the event of backup and restore. Therefore, it is critical that the database contents are securely backed up. In case of failure, the respective systems are deployed using the option of selecting an installation type that relies on an existing database.

5.1.1 Archived Data

ADSS Server implements a sophisticated archive service for all transaction, event and operational logs. This ensures the database does not bloat. The archive services can be configured per service as opposed to system wide, and operate on schedule, or number of records.

If this data must be kept then ADSS Server can be configured to digitally sign archived. This information must be kept safe is needed. At a future point, the log files can be imported into ADSS Server and it will verify the integrity of the information.

5.2 Architecture Notes

The server machine on which the ADSS Server is going to be deployed must be properly planned to handle concurrency needs.

5.2.1 Load Balancer / Traffic Manager

ADSS Server implements a sophisticated archive service for all transaction, event and operational logs. This ensures the database does not bloat. The archive services can be configured per service as opposed to system wide, and operate on schedule, or number of records.

5.2.2 Database

ADSS Server and ADSS Server Gateway instances should have their own respective unique database instance or cluster. The database is shared by all ADSS Server instances in that cluster. This ensures all nodes have access to the same respective configuration information as their peers.

ADSS Server instances utilise the same database information to not only share information but also to store common transient data. That is, data such as CRLs that are retrieved and ingested on a regular, periodic basis, e.g. CRL nextUpdate.

5.2.3 ADSS Server Master Slave Concept

ADSS Server Core and Console operate in a master slave scenario for high availability. This ensures that tasks are not repeated for efficiency (CRL retrieval and ingestion for example) and the two modules are fault tolerant. For further information see:

High Availability (ascertia.com)

Note ADSS Server Service modules act in standalone mode and multiple modules are access via a load balancer. This is because generally these modules serve synchronous requests from business clients.

For ADSS Server deployments all three ADSS Server modules, i.e. Console, Core, and Service, are deployed on one host as shown above.

There is no limit to the number of Slave instances for an ADSS Server cluster set-up.

5.2.4 Dependencies

The only dependency of ADSS Server is the database. Even the HSM is optional, and ADSS Server can function entirely with software-based keys. However, this is not recommended for production systems.

5.3 Session Management

ADSS Server operates with synchronous calls. With each call the client is identified, authenticated if required, and the request answered. There is no concept of sessions in the traditional sense for clients. Any failure to a Service module will result in a loss of the transaction and the requesting client must repeat the call or send it to another node.

Administrator access to the console does have the concept of sessions and the time out of these is controlled via the **Global Settings** menu of the Console.

6 Key Management

ADSS Server can manage all aspects of key and certificate lifecycle management. This is for both clients and internal infrastructure elements. Infrastructure keys are used to secure communication with ADSS Server, whereas clients use the keys and certificates for signing and verification operations.

ADSS Server has Key Manager and Certification Service modules. These dictate where ADSS Server sends certification signing requests for a client or administrator. In simple terms, this setting defines the location of ADSS Server Certification Service, which in turn dictates the final issuing CA – either internal ADSS Server or external managed CA. Note the Certification Service also dictates where the key pair is generated. It does this by referencing a cryptographic profile of **Key Manager**.

ADSS Server Certification Service defines the issuing CA (either internal ADSS Server CA or external CA). Note multiple profiles can exist and hence ADSS Server can utilise many issuing CAs.

6.1 SAM Service

Where the SAM Service is used within an appliance then as per certified mode, all signing keys are Assigned Keys and stored securely either in the appliance-bundled HSM or appliance-bundled database.

For non-certified mode signing keys are stored either on the standard PKCS#11 HSM, or ADSS Server database.

7 Local Signing & Go>Sign Desktop

ADSS Server offers considerable flexibility in its configuration to cope with a variety of business application and process requirements. One of these is local signing using smart cards or similar hardware devices. In such instances, ADSS Server relies on Go>Sign Service and Desktop to facilitate communication between itself and the signature creation device, which would otherwise not be possible.

7.1 Java Dependency

Prior to Go>Sign Desktop, Ascertia offered Go>Sign Applet. This was, of course, a Java based solution that used applets and required end users to deploy and run Java Plugin in their respective web browsers. Go>Sign Desktop, although Java based, is a client-side application and does not have any reliance on Java in the web browser. Therefore, ADSS Server fully supports client-side signatures when using modern web browsers such as Edge, and later version of Chrome for example.

Go>Sign Applet relied on NPAPI, which is a browser technology that Java Applets required to function in the end user browser. Go>Sign Desktop runs on the end user machine and not in the web browser.

Note Ascertia no longer offer Go>Sign Applet to new customers.

7.2 End-User Experience

Fortunately, ADSS Server ensures the user experience is no different when using local signing as opposed to central, server side. That is, ADSS Server interaction is the same with the only extra step required for the user to select the appropriate certificate and unlock their respective signature creation device by supplying the required PIN, passphrase or similar.

7.3 Go>Sign Desktop

Ascertia Go>Sign Desktop is a lightweight client-side application that allows ADSS Server to communicate with a local hardware signature creation device. Working in conjunction with ADSS Server the client application communicates with the given hardware device using either CNG/Keychain (if the hardware device provides such an interface) or using the vendor specific PKCS#11 library.

ADSS Server communicates with Go>Sign Desktop using JavaScript embedded in HTML 5 pages of ADSS Server. This JavaScript is supplied by the Go>Sign Service in real time for use in the end user browser.

7.3.1 Integrated Solution

It is possible to use a combination of the hardware vendor middleware and ADSS Server for local signatures. In such scenarios (UAE and Belgium eID cards) Ascertia worked with the hardware/middleware vendor to produce an integrated solution whereby Go>Sign Desktop is not required. For these use cases, a bespoke set of JavaScript libraries was produced that allows ADSS Server to communicate directly with the respective middleware client.

7.3.2 Go>Sign Service

Go>Sign Desktop relies on ADSS Server Go>Sign Service. This is the interface between the client application and essentially, ADSS Server backend (ADSS Server).

The interaction between Desktop and Service is depicted here:

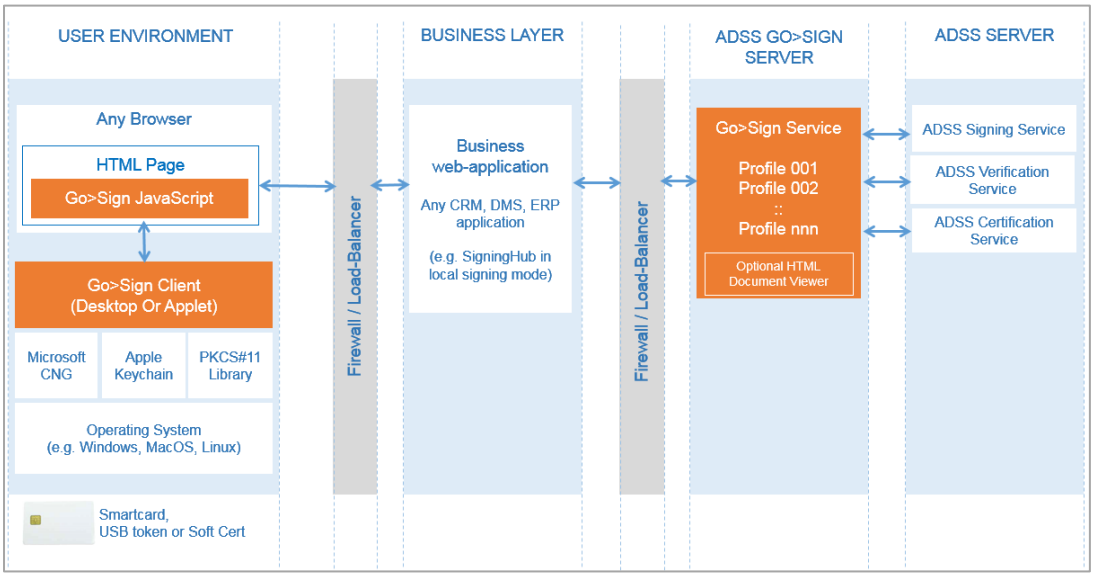

Figure 12 – Go>Sign Local Signing Workflow

In the above scenario, the business web application that is referred to can be ADSS Server as indicated.

7.3.3 Tomcat AJP

ADSS Server is a Java EE application that is deployed with three Tomcat instances. To support local signatures using hardware signature creation devices and Go>Sign Desktop, ADSS Server uses a Tomcat AJP Connector, deployed in a DMZ hosted web server, along with ADSS Server itself.

The connector is a standard implementation of AJP, supporting reverse proxy functionality along with URL rewrites if required.

The AJP Connector is one way to ensure there is no requirement to expose ADSS Server Go>Sign Service directly online. That is the communication flow from the user browser is always to a DMZ hosted web server that hosts the AJP Connector. The AJP Connector is responsible for the communication to backend ADSS Server Go>Sign Service.

A second way is to deploy ADSS Server in Gateway or Proxy mode and serve Go>Sign Service from the public facing server.

7.4 Go>Sign Desktop Ports & Protocols

For local signing operations Go>Sign Desktop is invoked by JavaScript, with instructions for signature from CNG/Keychain CSP implementation or PKCS#11 library from the hardware vendor. Go>Sign Desktop listens on ports 8782 for HTTPS secure traffic.

7.5 Security

For security reasons Ascertia can supply an Access Control List (ACL) as part of the Go>Sign Desktop package. This is signed with the rest of the package to protect against changes. The ACL is used to ensure only known hosts can invoke requests on the Go>Sign Desktop client, e.g. web.signinghub.com.

7.6 Traffic Flow

The following diagram shows the process flow when using Go>Sign Desktop in conjunction with ADSS Server to create digital signatures using a locally held hardware device:

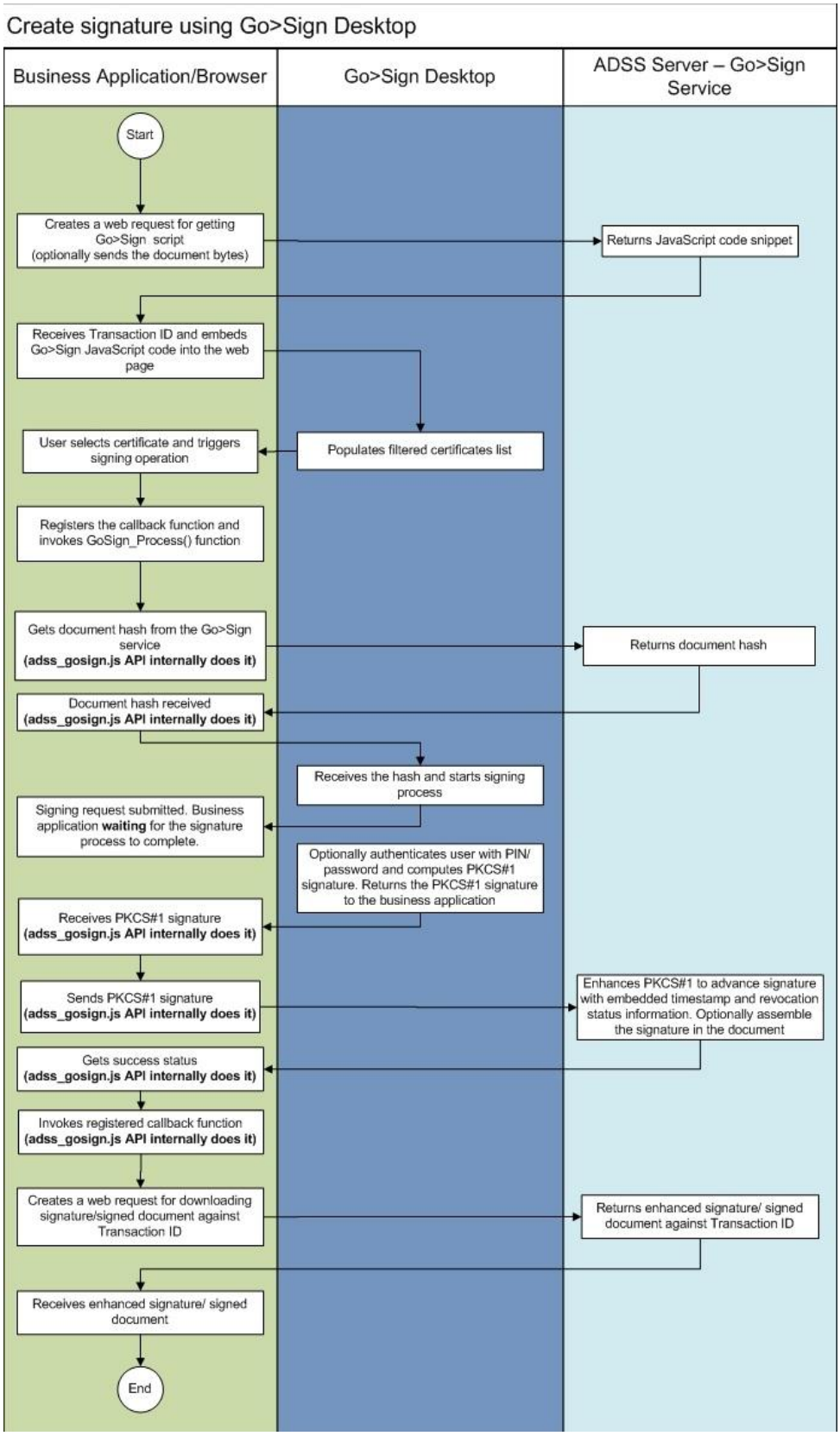

Figure 13 – Go>Sign Desktop & Service Process Flow

8 Security

ADSS Server now offers Virtual CSP (VCSP) support for Windows Desktops to support remote signing, i.e. where keys and certificates are held centrally. It works in conjunction with the VCSP Plugin installed locally and enables any Microsoft or third party CAPI/CNG application to conduct remote signing seamlessly. VCSP has been tested for remote signing with a range of applications including Microsoft Word, Outlook and Adobe Acrobat®.

ADSS CSP Service provides the capability to manage users and list their certificates. It provides the required API interfaces to manage users, certificates, handles signing requests & getting the signed hash (i.e. PKCS#1 signature) and their current statuses.

Virtual CSP is a lightweight client application deployed on the desktop. The user experience is the same as with any local hardware: they elect to sign and a document and are presented with a list of certificates by Microsoft CAPI/CNG. However, instead of requiring hardware, the request is sent to ADSS Server CSP Service for processing and signing. The signed blob is returned to the calling application and the document format is retained.

The architecture is as shown here:

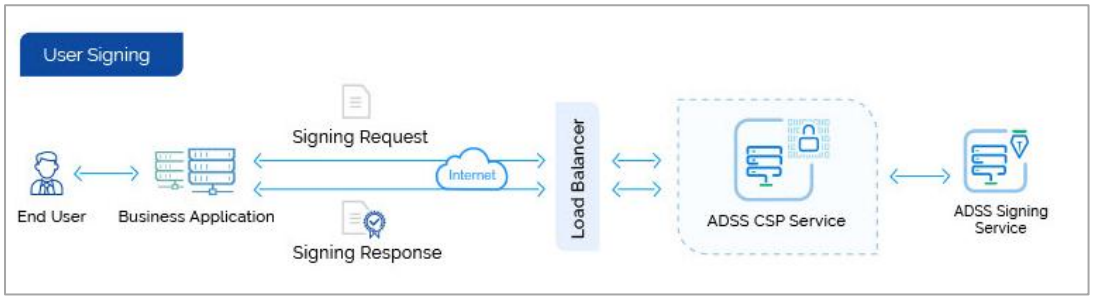

Figure 14 – Go>Sign Desktop & Service Process Flow

The CSP Service is another standard ADSS Server module and requests are handled in the same way as for any Service module.

*** End of Document ***