SigningHub

Architecture & Deployment

ASCERTIA LTD

FEBRUARY 2018

Document Version- 1.0.1.0

CONTENTS

FIGURES

TABLES

1 Executive Summary

This document describes the architecture and deployment scenarios for SigningHub Enterprise. This includes the dependency on 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 also covered such as database and HSM (Hardware Security Module). However, specific configuration of these is outside the scope of the document and the reader is advised to 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.

1.1 SigningHub Architecture Overview

SigningHub is a .NET application that runs on Windows Servers. SigningHub provides user management, document and workflow management, auditing, optional billing. It uses Ascertia's other strategic product, ADSS Server, to provide all the 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.

1.2 Security

To ensure optimal security SigningHub encrypts all documents using AES (Advanced Encryption Standard) 256-bit key as it receives them and before it stores them in the database, (both SigningHub and ADSS Server support database encryption). Each key is unique to the SigningHub deployment and is generated in real time during deployment. To provide further security SigningHub also performs all hashing of a document so that any cryptographic operation performed by ADSS Server is only performed on the hash rather than the full document. This allows ADSS Server to be run as a managed service if needed and this option is used by some partners.

SigningHub generates a unique deployment Document Encryption Key (DEK) per deployment to secure document data. SigningHub offers the capability to utilise an HSM to protect the DEK, using a further Key Encryption Key (KEK) generated and protected by the HSM. This is done via ADSS Server and SigningHub itself has no direct communication with the HSM.

SigningHub 7.3 saw a major change in the way it handles document storage. In this, and later versions, all documents, and associated workflow evidence reports are stored in a physical file location. The location of which is any local of nominated Universal Naming Convention (UNC) address. When using a UNC location, it is important to ensure the correct permissions are set to allow SigningHub to read and write from said location. The installation document covers this topic.

Database encryption is supported to further strengthen the security of any deployment. This applies to both SigningHub and ADSS Server databases respectfully.

1.3 Presentation Layer Overview

SigningHub uses Web 2.0 and HTML 5 technology to provide a quick and easy way for people to review and sign documents. SigningHub can easily work with internal or external users and has a wealth of options to ensure that it is quick to install and works within existing process flows, with existing web and mobile applications, with multiple existing authentication methods and with existing PKI (Public Key Infrastructure) signing keys and associated certificates.

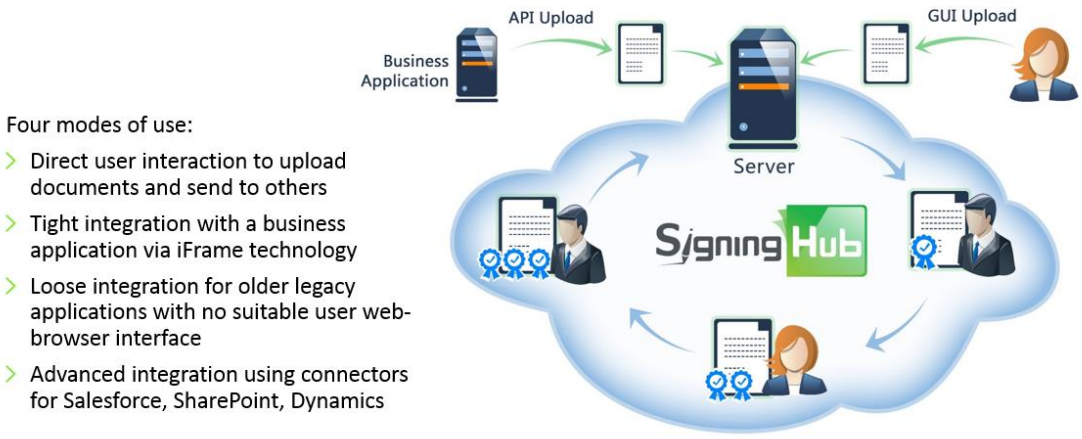

Figure 1 SigningHub Interaction Overview

The diagram shows that SigningHub can be driven both by human interaction and by RESTful (REpresentational State Transfer) API (Application Programming Interface) calls from any business application. The full API reference is available here:

http://manuals.ascertia.com/SigningHub-apiguide/default.aspx#pageid=welcome.

1.4 Deploying SigningHub

SigningHub 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.

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 all 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 or different site. This latter option allows SigningHub to be deployed on premise whilst all the user keys are held within an ISO 27001, SSAE 16, SOC 2 datacentre.

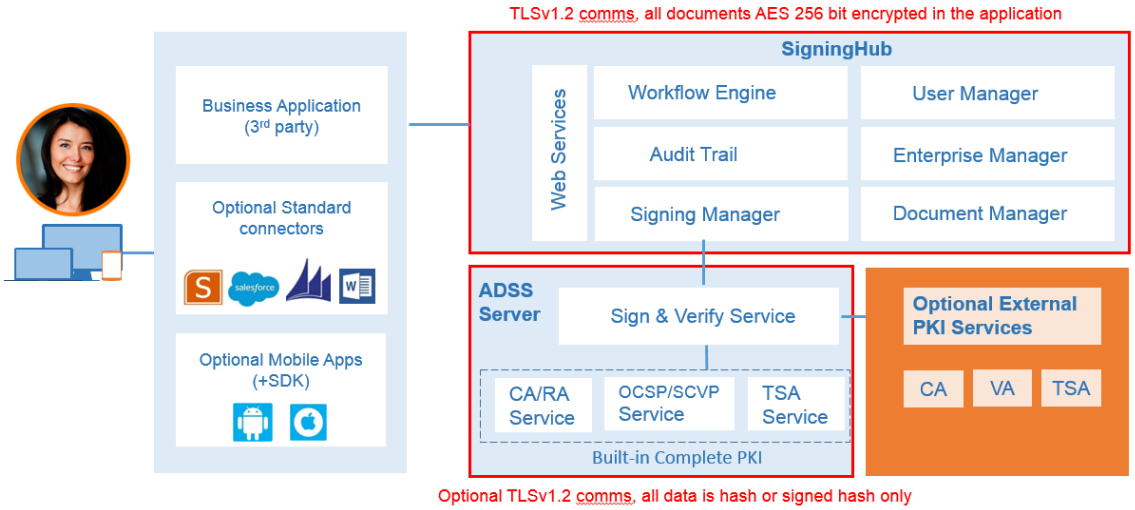

Figure 2 SigningHub & ADSS Server Components

1.4.1 Scenarios

Ascertia categorise deployment of SigningHub into four main types: -

- Internal Service: On premise SigningHub deployment used exclusively by employees of one organisation.
- External Service: On premise SigningHub deployment used by employees of one organisation and clients of the organisation.
- Hosted Private Service: Cloud hosted service to cover scenarios 'Internal and External Services defined directly above).
- Commercial Managed Service: Partner or customers hosting SigningHub to paying customers from any organisation.

In terms of features all are available with respect to SigningHub and ADSS Server.

1.4.2 Resilience & Scalability

SigningHub is well proven to offer a robust and resilient platform. Ascertia's SigningHub cloud service (https://web.signinghub.com) runs on an Azure instance in Northern Europe and has been running continuously with no unplanned downtime after several years of operation by many thousands of users. Other Ascertia partners around the world have similar experiences.

Similarly, the underlying ADSS Server has a long history of running 24/7 without intervention. High availability services can be ensured by using two production servers running SigningHub with a back-end database cluster offering resilient information services to deliver a service of 99.9% or better. This will be dependent on the IT equipment and environment, staffing, test procedures etc.

The disc storage and database management system are vital for high availability. Separate SigningHub 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.

1.4.3 Typical Deployment

The following diagram depicts a typical deployment of SigningHub: -

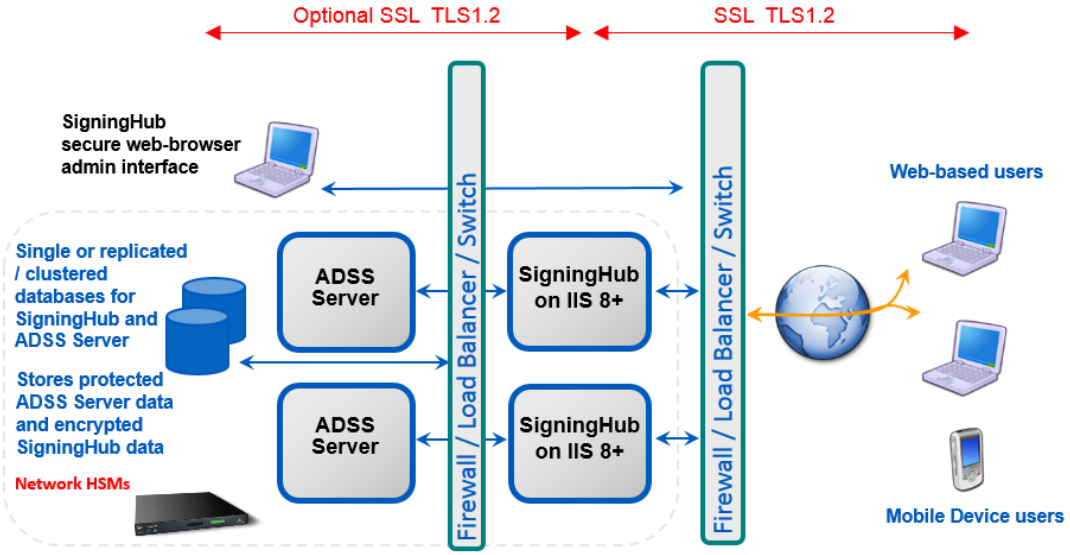

Figure 3 SigningHub Typical Deployment

It is expected that an EV-SSL (Extended Validation Secure Socket Layer) certificate is used and TLSv1.2 is configured within IIS.

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

Note both SQL Server standalone, and Azure SQL Database are supported database repositories for SigningHub. Furthermore, IIS 10 is supported.

For a complete list of supported third party elements refer to the installation guide.

1.5 Database High Availability

SigningHub and ADSS Server require their own respective databases, and each relies on the database vendor to provide high availability, fault tolerance and disaster recovery to ensure their data is always available. Therefore, both components rely on a cluster, instance or similar single connection point, and rely on the database vendor to manage everything else in terms of availability.

Features employed by SQL Server and Azure SQL Database such as Always On for example, and therefore implicitly supported.

1.6 HSM High Availability

As with databases ADSS Server relies on a single connection point provided by the HSM vendor, and leaves all fault tolerance, high availability, and disaster recovery to the HSM vendor technology. For example, Gemalto SafeNet HSM client load balancing and traffic management.

2 Architecture

For a typical SigningHub deployment (inclusive of ADSS Server) as depicted above in 1.4.3 Typical Deployment, the architecture would look similar to the following: -

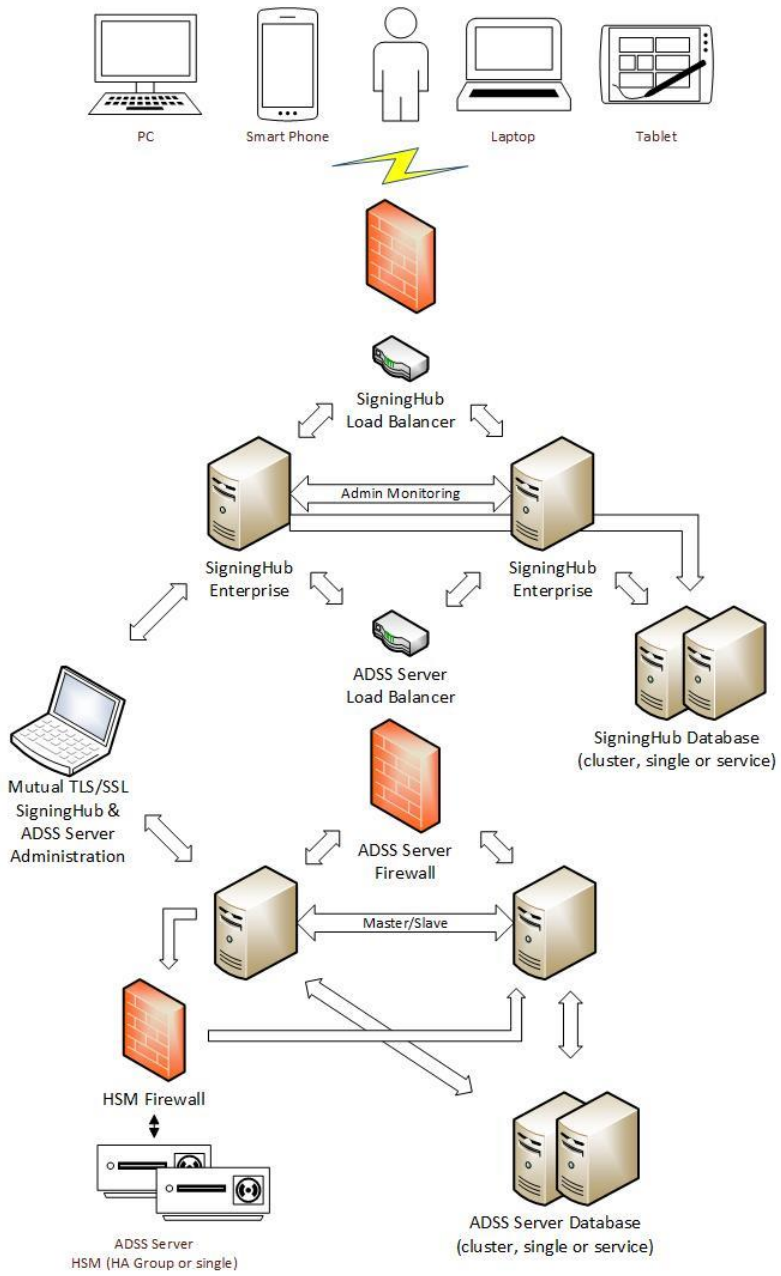

Figure 4 SigningHub Typical Deployment Architecture

Note the only communication between SigningHub instances is the Admin service that keeps a check on all known instances in the cluster. That is, Admin, Web, API, Mobile Web, Office, Demo, and Core web sites from all known other SigningHub instances. This is purely for a monitoring perspective and to alert if a service is not functioning or not available. This list is available for viewing the administration console of SigningHub as shown here: -

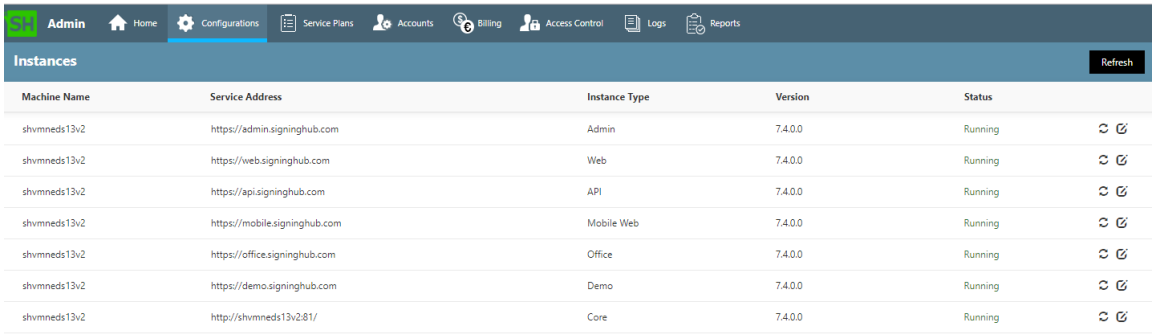

Figure 5 SigningHub Administration Console Instance Monitoring Example

There is no concept of master/slave within SigningHub nodes, nor active/passive operation. All nodes, when functioning, are active and can accept requests.

2.1 SigningHub & ADSS Server Communication

SigningHub is the digital signature workflow engine whilst ADSS Server provides all the cryptographic functionality required. SigningHub 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, SigningHub performs the document hashing and formation of the final signed PDF or Word documents1. Therefore, bandwidth requirements between SigningHub and ADSS Server are minimal for server-side signing.

For example, using the default SHA-256 hashing algorithm the messages sent from SigningHub to ADSS Server are 32-byte blobs for signature. However, for local signing bandwidth requirements are influenced by the size of the document to be signed. As this is outside the control of SigningHub to a point, requirements will vary. SigningHub service plans can restrict the maximum size of the document accepted to prevent extremely large document requests. This model ensures the two elements can exist in entirely separate locations, and that ADSS Server can support multiple SigningHub instances. These are the specific services used by SigningHub: -

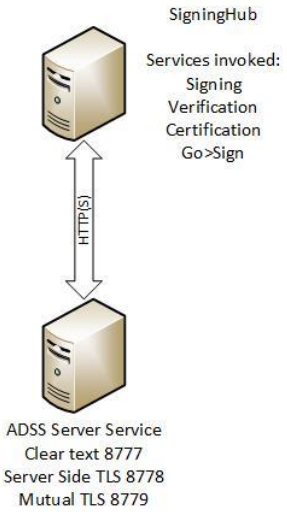

Figure 6 SigningHub & ADSS Server Communication

[1] *Applies to server side signing operations only. Local signing operations rely on ADSS Server Go>Sign Service to perform hash operations.*

2.2 SigningHub on IIS

SigningHub elements are all deployed as IIS web sites, including Core module (up until 7.3 this was a Tomcat instance deployed as Windows Service). This is the communication architecture for SigningHub alone: -

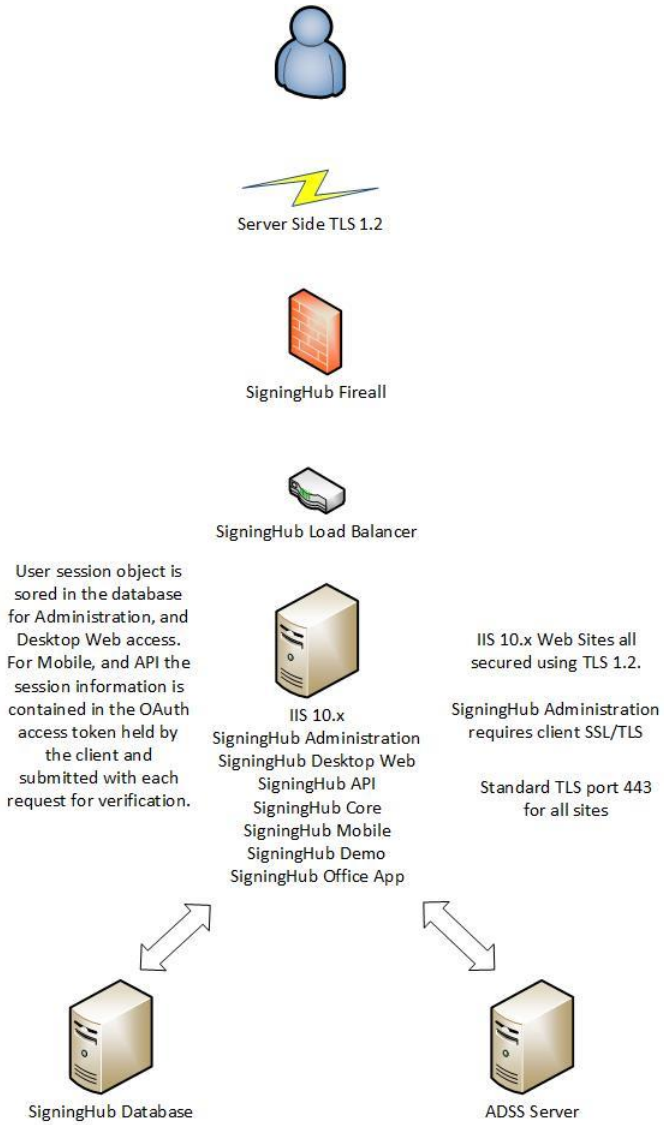

Figure 7 SigningHub IIS Components

2.3 SMTP Dependency

SigningHub uses email as the primary notification medium. User registration, and all notifications are sent via SMTP. Hence it is a critical part of the architecture and deployment. Communication is standard SMTP: -

Figure 8 SigningHub SMTP Dependency

Standard SMTPS is supported if required.

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. Hence the architecture is the same as SigningHub above: -

Figure 9 ADSS Server SMTP Dependency

2.4 System Administration

Both SigningHub and ADSS Server administration consoles are protected by server-side TLS and require client TLS authentication to access the site. The certificate is mapped to a logical identity in both systems once IIS or Tomcat has performed the actual authentication. In addition, a password can further supplement the client TLS authentication. This ensures it is not possible to restart a session when software-based keys are used and cached by the web browser.

ADSS Server is available directly or via AJP Connector deployed in SigningHub Desktop Web web-site. Herewith the architecture for this: -

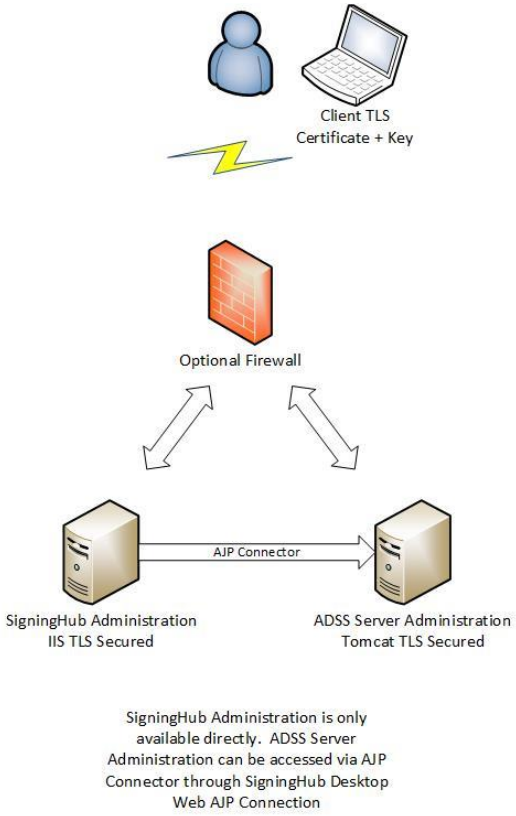

Figure 10 SigningHub & ADSS Server Administration

2.5 ADSS Server Tomcat Connectors

ADSS Server is a Java 8 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 for local signing in SigningHub. Here are the connectors and communication flows: -

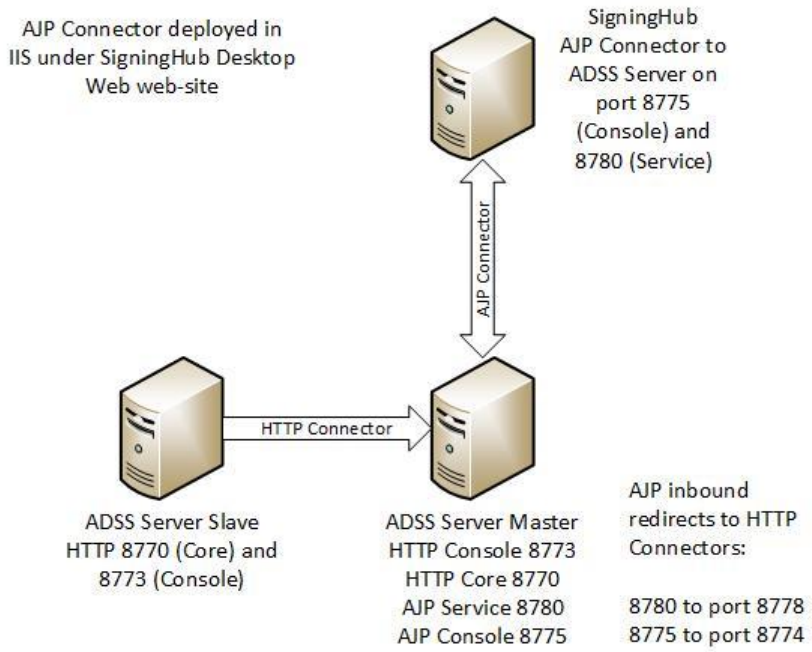

Figure 11 ADSS Server Tomcat Connectors

2.6 Local Signing using SigningHub

For specific use cases of local signing employing locally held signature creation devices such as USB Tokens or smart cards SigningHub relies on Go>Sign Desktop and Service. The interaction is covered in more detail in section 6 Local Signing & Go>Sign Desktop. For introduction, this is how the combination of SigningHub, ADSS Server, and Go>Sign components communicate to support the use case: -

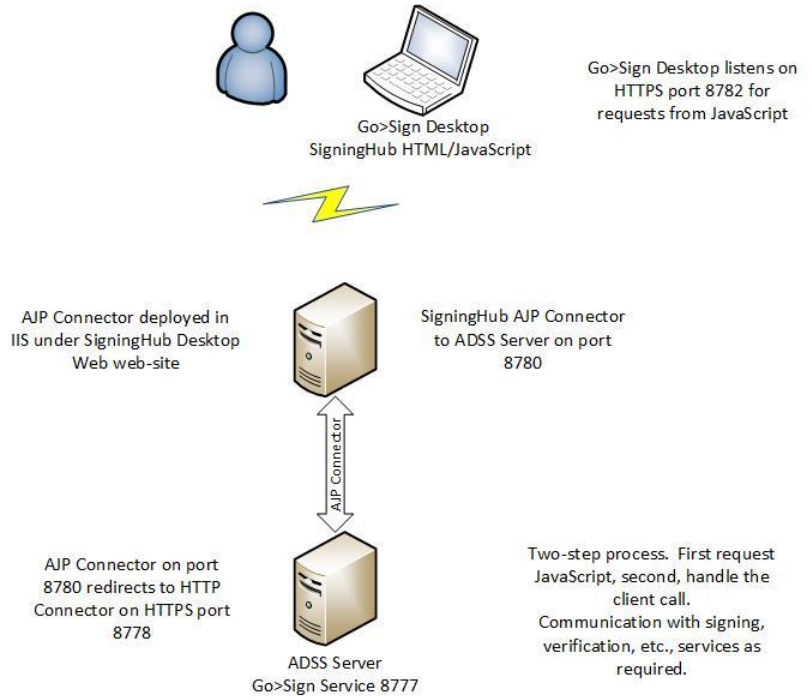

Figure 12 Local Signing

3 Communication

This section details the protocols, ports and flows of communication that occur within a SigningHub deployment. SigningHub is comprised of SigningHub and ADSS Server. Each provides numerous access points to support services.

3.1 SigningHub Modules

SigningHub itself is made up of several components. These are as follows: -

- SigningHub Administration: Web based console for system management and configuration.
- SigningHub Desktop Web: Main user interface for desktop users.
- SigningHub API: REST architectural style API interface.
- SigningHub Website Integration Demo: Fully implemented and configured REST architectural style API sample application.
- SigningHub Mobile Web: Main user interface for mobile web users. Geared towards browsers on mobiles but contains all the same features and functionality as SigningHub Native Apps.
- SigningHub Office App: IIS module to specifically support SigningHub Microsoft Word App. That is, documentation preparation and signature from within Word.
- SigningHub Core: Management module that offers services to support all other modules. For example, sending of email notifications, physical document management lifecycle.

Of the elements mentioned above, all are IIS based.

In addition to the above server-side modules there are SigningHub Apps for iOS and Android, and native apps for SharePoint, Dynamics CRM, Word, and Salesforce.

3.1.1 Installation & Deployment

The modules defined above are not all mandatory. SigningHub modules should be deployed to fit the business scenario. The mandatory modules required to deploy and operate SigningHub are: -

- SigningHub Administration
- SigningHub Desktop Web
- SigningHub Core

SigningHub Mobile is not mandatory but is essential to provide the best user experience on mobile devices. Without it documents will not render correctly and some of the features required by end users, unavailable or inaccessible. If Mobile is required then API becomes mandatory because of the dependency of the Mobile Apps on the APIs.

3.1.2 User Authentication

SigningHub always authenticates a user who attempts to access the dashboard/desktop. The same applies to mobile web use case, mobile native apps, and browser based access. Additional authentication can be enforced at the time of signature for server side held signing keys. This provides greater assurance for the signature.

Authentication types are controlled via a user's role and respective settings. The role aspect allows an enterprise administrator to control a user base. It is possible to set an authentication type for server side signing for both desktop and mobile apps. This allows greater granularity than one broad setting.

With respect to electronic signatures there is no user authentication because the user, essentially, does not exist in SigningHub. Therefore, there is no associated authentication profile.

However, it is still possible to enforce authentication of the signatory by employing document access protection via password or OTP via SMS, or simply OTP via SMS at the point of signature.

SigningHub supports the following authentication methods: -

- SigningHub ID (user name and password)
- Active Directory
- Active Directory Federated Services
- Office 365
- Entrust IdentityGuard
- AET Consent ID
- Verisec Freja Mobile
- Verisec Freja eID
- Salesforce
- SAML v2.0 IdP
- LinkedIn
- Google
- Ubisecure
- SSL Client Authentication (additional password requirement can be added)
- Remote Authorised Signing

3.1.2.1 SAML

SigningHub is a fully compliant SAML v2.0 SP (Service Provider). Therefore, any SAML v2.0 IdP (Identity Provider) can be used for user authentication.

3.1.3 SigningHub External Interfaces

All SigningHub services are exposed via HTTP(S). The receiving host is IIS and not proprietary SigningHub.

IIS is also responsible for the security aspect in terms of confidentiality (TLS/SSL) and in the case of SigningHub Administration Console, mutual TLS/SSL implementation. Once the IIS has successfully authenticated the administrator, SigningHub uses the full certificate to match against a known, and active, administrator, and thus allow access. Note SigningHub performs the password validation if configured.

The REST architectural style API is implemented using ASP .NET Web API, but still within IIS.

3.1.4 SigningHub External Dependencies

3.1.4.1 Database

SigningHub must have access to a suitable database. Currently this is SQL Server or Azure SQL Database. In both cases SigningHub maintains and manages its own database connection pool and interaction. Underneath, this uses Dapper ORM (Object-Relational Mapping) technology. The connection pooling is handled by Microsoft SQL Server JDBC drivers.

Only the Admin, Web, Core, and API components of SigningHub communicate with the database.

3.1.4.2 SMTP

A key part of SigningHub functionality is document workflow and to implement this the system relies on emails, specifically SMTP. Therefore, a suitable SMTP server must be available for use. Reliance for SigningHub is the connection details of course.

3.1.4.3 External Authentication Providers

External authentication providers such as SAML v2.0 IdP must be accessible if used to identify users and authenticate them. Note SigningHub is a SAML v2.0 compliant SP.

The default authentication method enforced by SigningHub is user name (email address) and password, which does not rely on an external provider.

3.1.4.4 Document Libraries

Dropbox, Google Drive, or Microsoft OneDrive if required for user document library. Note access to cloud drives is allowed via a user's service plan in SigningHub.

3.1.4.5 HubSpot

If using HubSpot for marketing and statistics access is required.

3.1.4.6 SMS Provider

Documents and signature positive act of intentions can be protected using OTP via SMS. For such cases access to Twilio or Clickatell services is required.

3.1.4.7 Payment Processor

SigningHub supports card payments through WorldPay and Stripe/Taxamo.

3.1.4.8 Push Notifications

SigningHub supports Push notifications to Android and iOS mobile devices. The notifications are only sent to users who have a SigningHub account and have the Mobile App installed on their respective devices.

3.1.5 SigningHub Ports & Protocols

Note there is no HSM interface from SigningHub. ADSS Server is the only component that communicates with an HSM.

Port[2]	Protocol	Initiator	Target	Description
443	HTTPS over TCP/IP	Administrator web browser	SigningHub Administration	Mutual TLS/SSL to access SigningHub Administration Console.
443	HTTPS over TCP/IP	User web browser. Third party business applications that use REST API tight integration.	SigningHub Desktop Web	Server-side TLS/SSL to access SigningHub Desktop Web for standard use. This includes the SigningHub viewer to

[2] *All ports are default and configurable during deployment.*

		SigningHub Natives Apps for Salesforce, SharePoint, Dynamics CRM, and Word.		display signature panel for review, signature, and document preparation.
443	HTTPS over TCP/IP. JSON over HTTPS.	SigningHub Mobile App. SigningHub Natives Apps for Salesforce, SharePoint, Dynamics CRM, and Word. User mobile browser. SigningHub Mobile SDK. Third party business applications.	SigningHub API	Server-side TLS/SSL to access SigningHub API.
443	HTTPS over TCP/IP	User web browser.	SigningHub Demo	Server-side TLS/SSL to access SigningHub Demo REST API Application.
443	HTTPS over TCP/IP	SigningHub JavaScript used during local document view and signature operations.[3]	AJP Connector to Go>Sign Service	Go>Sign Service communication that relies on AJP Connector to support local signatures.
443	HTTPS over TCP/IP	User web browsers on mobile devices (not to be confused with SigningHub Mobile Apps).	SigningHub Mobile Web	Server-side TLS/SSL to access SigningHub for mobile optimised web site.
443	HTTPS over TCP/IP	SigningHub Microsoft Word App.	SigningHub Office App	Server-side TLS/SSL to access SigningHub Office App.
81	HTTP over TCP/IP	SigningHub Admin nodes hosted by IIS.	SigningHub Core	Internal use only; not exposed outside of SigningHub host.

Table 1 SigningHub Ports & Protocols of Communication

3.2 ADSS Server Modules

ADSS Server is a Java 8 EE application deployed in three separate Tomcat instances. It consists of many modules that SigningHub 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.

[3] *Note the actual implementation relies on Go>Sign Desktop or Applet. This is required when signing the documents using the credentials held locally in MSCAPI, Mac Keychain, Smart Card, Token or an eID card. The Go>Sign Service profile configuration dictates the usage of Desktop or Applet. It is a mutually exclusive setting. There is no requirement to operate both configurations, and Ascertia recommends Desktop for the majority of cases, and default option.*

3.2.1 ADSS Server & SigningHub Dependency

SigningHub must have access to ADSS Server to function. It is a mandatory component for any deployment of SigningHub. However, ADSS Server can function as a standalone product in its entirety. Regardless of the architecture deployed, all ADSS Server services can be accessed independently of SigningHub. For example, one ADSS Server instance can support a full SigningHub deployment and provide a separate time stamping service directly to business clients.

In addition, this means all signature formats are available by directly accessing the signing service of ADSS Server from any business client. These are extensive and details here: http://manuals.ascertia.com/ADSS-Admin-Guide/default.aspx?pageid=types.

Go>Sign Service and Desktop can be used independently of SigningHub and the supported signature formats for this are:

http://manuals.ascertia.com/ADSS-Admin-Guide/default.aspx?pageid=supported_signature_type.

3.2.2 ADSS Server External Interfaces

All external interfaces of ADSS Server can be summarised as follows: -

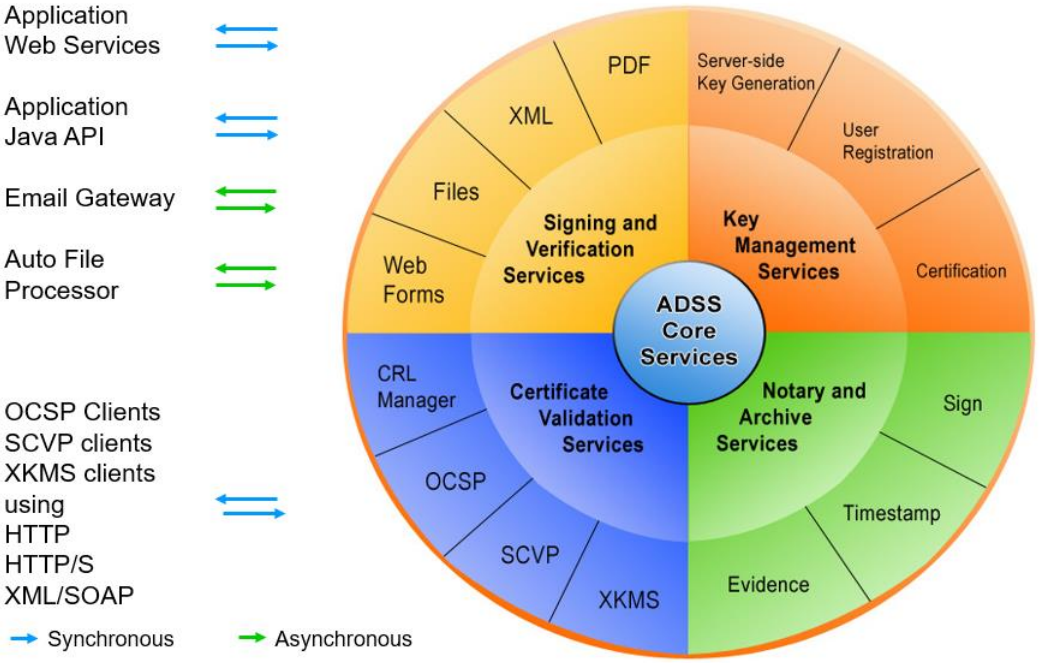

Figure 13 ADSS Server External Interfaces

SigningHub is a client of ADSS Server, and communicates directly with the services defined above.

ADSS Server, where relevant, supports and adheres to IETF and OASIS standards. OCSP, SCVP, and TSA (RFC 6960, 5055, and 3161 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.

3.2.3 ADSS Server External Dependencies

3.2.3.1 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).

3.2.3.2 SMTP

If required for alert notifications to designated administrators.

3.2.3.3 SMS Provider

If required for alert notifications to designated administrators.

3.2.3.4 SNMP Service

If required for alert notifications to designated administrators.

3.2.3.5 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.

3.2.3.6 NTP Service

Used for a time source for local TSA, and for verification that the local system clock is correctly synchronised to an accurate time source.

3.2.3.7 Revocation Information

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

3.2.3.8 External CA

If using an external CA for certificate issuance. Current supported CAs: Microsoft AD CS, PrimeKey EJBCA, GlobalSign EPKI, QuoVadis, and any other ADSS Server CA.

Note there is no current interface for VeriSign/Symantec issuing CA.

3.2.3.9 External TSA

If using an external TSA for time stamp operations.

3.2.3.10 Hardware Security Module

If using an HSM for secure key generation, use and storage. Interface is based upon PKCS#11 standard for direct communication, and RESTful API for Microsoft Azure Key Vault. MSCAPI/NCG is another option.

3.2.4 ADSS Server Ports & Protocols

Port[4]	Protocol	Initiator	Target	Description
8773	HTTP over TCP/IP	ADSS Server Console – slave instance	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	SigningHub AJP Connector	ADSS Server Console	Access to ADSS Server Console via AJP Connector deployed on SigningHub IIS host. Redirects to port 8774.
8770	HTTP over TCP/IP	ADSS Server Core – Slave instance	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.
8777	HTTP over TCP/IP	ADSS Server Client. For example, SigningHub or AFP. SigningHub 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, SigningHub or AFP. SigningHub 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, SigningHub 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	SigningHub 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.

Table 2 ADSS Server Ports & Protocols of Communication

[4] *All ports are default and configurable during deployment.*

4 High Availability

Production deployments of SigningHub 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 host the solution.

4.1 Data

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

4.1.1 Document Storage

Documents and workflow evidence reports are stored on a local file system level, or any valid UNC address. These folders and files must be backed up to ensure they are not lost and can be retrieved in the case of emergency.

Note the IIS web site application pool of Desktop Web must have the configured user to allow permissions for this storage.

4.2 Architecture Notes

Ascertia recommends two channels of deployment. That is, traffic managed from the Internet through a load balancer to two channels of deployment.

4.2.1 Load Balancer

The load balancer to SigningHub has no application level specific requirements. This is because SigningHub relies on the database to store session information for Administrator and Desktop Web usage, and in the case of mobiles and mobile apps, the session information is held by the requesting client in terms of the access token, which is validated upon submission with each request.

Load balancer for traffic to ADSS Server instances need not be sticky session based either.

4.2.2 Database

SigningHub and ADSS Server instances require their own respective unique database instance or cluster. The database (for SigningHub) is shared by both SigningHub instances, and the same model applies to ADSS Server, i.e. ADSS Server database instance or cluster is shared by all ADSS Server instances. This ensures both instances [SigningHub or ADSS Server] have access to the same respective configuration information as their peers. For example, user settings for SigningHub and Signing, Certification and Verification profiles of ADSS Server.

See Figure 4 SigningHub Typical Deployment Architecture that shows the two independent database instances or clusters for both SigningHub and ADSS Server.

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.

4.2.3 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:

http://manuals.ascertia.com/ADSS-Admin-Guide/default.aspx?pageid=high_availability_settings.

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

4.2.4 Dependencies

Every SigningHub instance relies on an ADSS Server instance. Theoretically one ADSS Server can support multiple SigningHub instances but this deployment scenario introduces a single point of failure.

4.3 Session Management

SigningHub operates with a common concept of user sessions. Each session is uniquely identified and managed. Currently session management is memory where there is a single SigningHub instance and database for deployments with more than one node. Therefore, there is no synchronisation of session information across SigningHub instances.

If a SigningHub instance fails, there is no impact accept for less capacity. Document status, current signatures, etc., are all retained because that information is stored in the respective database.

For clients using REST architectural style API they manage the OAuth access token and presents this with each request. Hence a failure of a SigningHub instance means no loss of service or session. That is, the access token is verified by the SigningHub node that receives the request.

4.3.1 Cookies

SigningHub relies on cookies to hold session information/identifier for administration and Desktop Web. Underpinning this is the ‘true’ random number generator of .NET, which is used to produce the actual identifier placed in the session cookie. The session cookie allows SigningHub to identify the user session and thus provide appropriate access to a valid user. The session cookie lifetime can be controlled via SigningHub administration and a hard-logout enforced after the designated period of user inactivity.

The cookie implementation is industry standard and ties the cookie to the HTTP(S) session in the browser.

Secure cookies are supported and this is configured via standard IIS configuration.

4.3.2 Document Data

Document data is stored (encrypted using AES-256 bit key) on the local file system or UNC location, and end users only see an image of what they are signing. Thus, a loss of a SigningHub instance does not affect the current status of the document nor any actions already performed therein. In addition, the workflow status is maintained and ready once a new SigningHub session is established by the client or user.

5 Key Management

SigningHub has no concept of private signing keys nor their respective associated certificates. SigningHub relies on ADSS Server to manage this aspect and certificate lifecycle management.

SigningHub implements a concept known as Service Plans. Every user is assigned to one Service Plan and this plan defines the characteristics and features for any user assigned the plan. For example, a service plan defines the number of signatures, templates and workflows a user has access to, along with the maximum storage space and document size allowed.

Within a Service Plan a certification profile or connector is defined. This dictates where SigningHub sends certification requests for a user. In simple terms, this setting defines the location of ADSS Server Certification Service.

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.

A Service Plan allows an administrator to configure one or more Certification Profiles. This allows a user to choose the respective certificate used in a signing operation. This choice is available at signing time. As with any Certification Profile, if the SigningHub user has no registered certificate for that particular profile then a signing request will result in a new key pair generation and certificate issued from the configured CA for the Certification Profile in use.

5.1 SigningHub Certificate Request

A user must have a service plan defined. This is mandatory because key generation and certificate issuance is governed by the characteristics of the Service Plan. Therefore, key generation and certificate issuance requests are only issued for those users whose respective service plan defines server-side signatures, and at the point of signature, i.e. when the private signing key and associated certificate is required.

For users, whose service plan defines local signing only, there is a never an instance where a key generation and certificate issuance request will be made.

6 Local Signing & Go>Sign Desktop

SigningHub 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, SigningHub relies on Go>Sign Desktop to facilitate communication between itself and the signature creation device, which would otherwise not be possible.

Go>Sign Desktop & Service are elements of ADSS Server and can in fact work independently of SigningHub. However, in the context of SigningHub they operate seamlessly without making the user aware of this.

6.1 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, SigningHub 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.

6.2 End-User Experience

Fortunately, SigningHub ensures the user experience is no different when using local signing as opposed to central, server side. That is, SigningHub 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.

6.3 Go>Sign Desktop

Ascertia Go>Sign Desktop is a lightweight client-side application that allows SigningHub to communicate with a local hardware signature creation device. Working in conjunction with SigningHub 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.

SigningHub communicates with Go>Sign Desktop using JavaScript embedded in HTML 5 pages of SigningHub.

6.3.1 Integrated Solution

It is possible to use a combination of the hardware vendor middleware and SigningHub 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 SigningHub to communicate directly with the respective middleware client.

6.3.2 Go>Sign Service

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

The interaction between Desktop and Service is depicted here: -

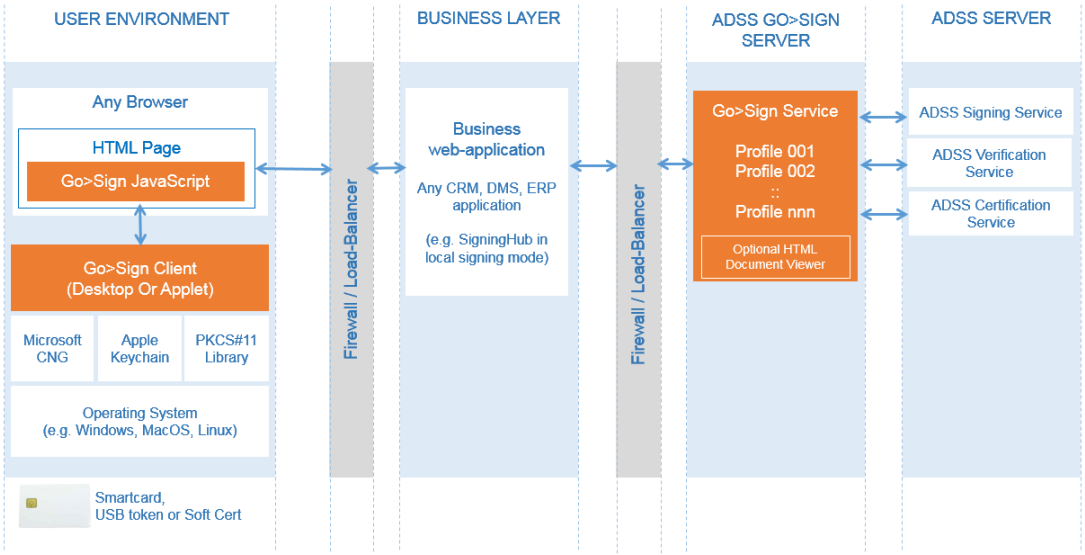

Figure 14 Go>Sign Local Signing Workflow

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

6.3.3 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, SigningHub uses a Tomcat AJP Connector, deployed in IIS along with SigningHub itself.

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

The AJP Connector is used 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 SigningHub IIS instance that hosts the AJP Connector. The AJP Connector is responsible for the communication to backend ADSS Server Go>Sign Service.

6.3.4 Access Control List

Go>Sign Desktop is distributed as a signed object/executable file. Within this Ascertia can include an Access Control List (ACL) of trusted IT addresses or DNS hosts from which to accept requests from. As this ACL is digitally signed by Ascertia prior to distribution it is only trusted if the integrity remains intact. With the ACL Go>Sign Desktop will only accept requests from known, trusted clients that appear on the ACL. This ensures it cannot be arbitrarily invoked, either mistakenly or for malicious purposes.

6.4 Go>Sign Desktop Ports & Protocols

For local signatures 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 port 8782 for HTTPS secure traffic.

6.5 Traffic Flow

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

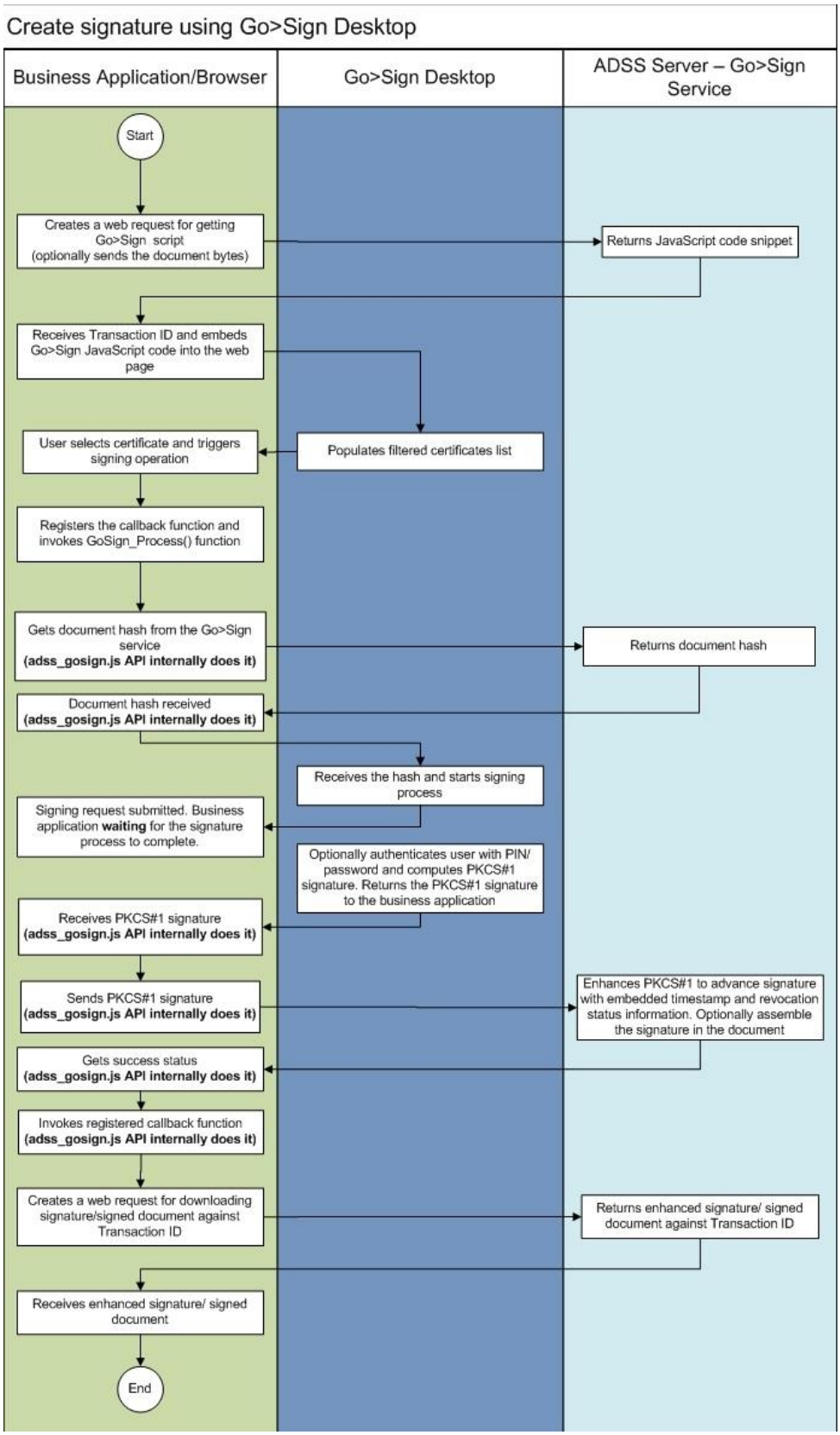

Figure 15 Go>Sign Desktop & Service Process Flow

*** End of Document ***