

Sign Live! cloud suite gears wp architecture

Oktober 2025

intarsys GmbH

Sign Live! cloud suite gears

wp architecture

Version 8.14

cloud suite gears

intarsys GmbH
Sign Live! cloud suite gears wp architecture
Version 8.14

All rights reserved
© 2019 intarsys GmbH
www.intarsys.de

Preface

- Author and company

This book has been provided by different authors from the development staff of intarsys GmbH.

- Trademarks

Wherever possible and where the authors were aware of a trademark claim, such designations are marked as trademarks in this book.

jPod is a trademark of intarsys consulting.

Sun, Java and JavaScript are trademarks of Oracle

Microsoft and Windows are trademarks of Microsoft Corporation.

- Who should read this book

This book provides both an overview of the product design and architecture and a reference for using the components and services.

So, this is the document for architects, developers and operators.

- Reviews and comments

We make constant efforts to improve our documentation and meet your requirements. Your comments are welcome and are a valuable resource for us.

Email support@intarsys.de

Website www.intarsys.de

Contents

| | |
|--|----|
| Preface | 5 |
| ▪ Author and company | 5 |
| ▪ Trademarks | 5 |
| ▪ Who should read this book | 5 |
| ▪ Reviews and comments | 5 |
| Contents | 6 |
| 1. Overview | 9 |
| 2. Reference architecture | 10 |
| 3. Communication scenarios | 11 |
| 3.1 Synchronous requests | 11 |
| 3.2 Asynchronous requests, front channel signalling | 12 |
| 3.3 Asynchronous requests, back channel signalling | 13 |
| 3.4 Asynchronous requests with multiple participants | 14 |
| 4. Network architecture | 16 |
| 4.1 Overview | 16 |
| 4.2 Firewall | 16 |
| 4.3 Proxies | 17 |
| 4.4 Load balancing | 17 |
| 4.4.1 Typical load balancer strategies | 17 |
| 4.4.2 Hash based load balancing | 17 |
| 4.4.3 Transparent load balancing | 18 |
| 4.4.4 Explicit load balancing | 18 |
| 5. gears | 19 |
| 5.1 Conversations | 19 |
| 5.2 Protocol specifics | 19 |
| 5.3 Builtin load balancing support | 19 |
| 5.4 Usage scenarios | 20 |
| 5.4.1 Synchronous requests | 20 |
| 5.4.2 Single instance gears | 20 |

| | | |
|-------|-----------------------|----|
| 5.4.3 | Asynchronous requests | 20 |
| 5.4.4 | Bridge | 22 |
| 6. | External References | 23 |

1. Overview

This book shows typical gears communication scenarios and the implications on typical system infrastructure and architecture.

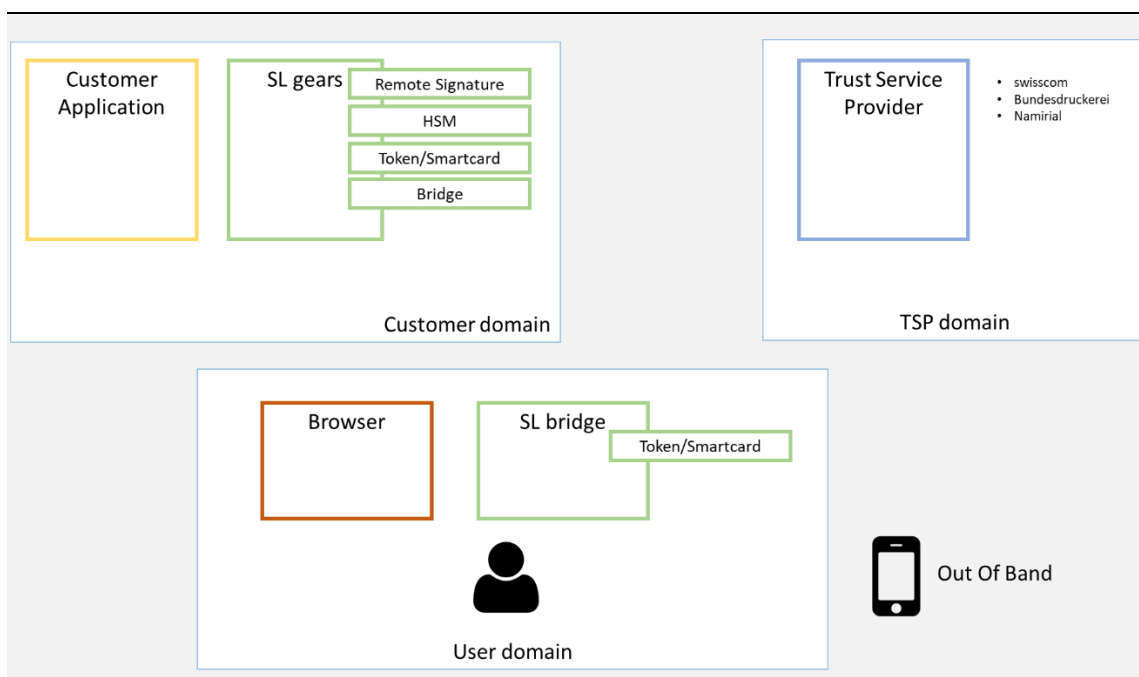
First, you should make yourself familiar with the different communication scenarios and be able to determine the one that applies to your installation.

Then you can determine the required configuration & setup you need in your infrastructure.

2. Reference architecture

This graphic illustrates the basic architecture of a gears installation that is used throughout the documentation set.

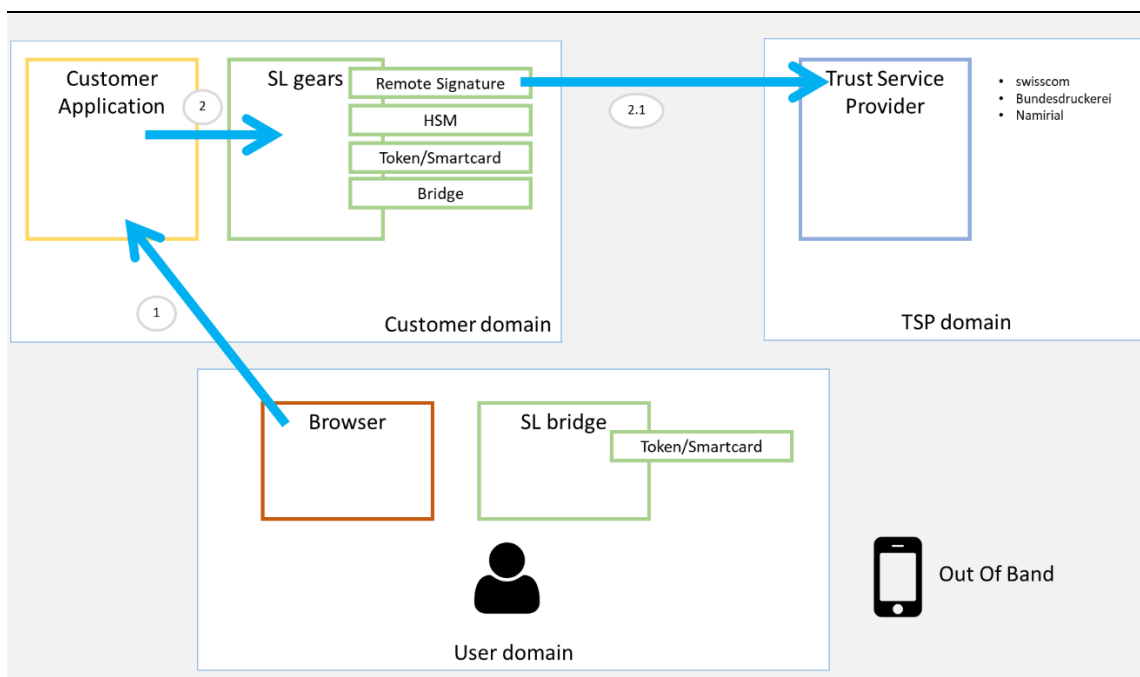
All infrastructure devices (like proxies, load balancer) are abstracted, these will be covered in a special chapter.



3. Communication scenarios

3.1 Synchronous requests

All communication with gears start with a request in the form shown below. Optionally we have outgoing requests to 3rd party providers.



gears replies with a "ReplyStage", an indirection that comprises the current state of the conversation you just started with gears. This course is **always** part of the interaction with gears.

In the simplest form, your client simply sends requests to gears that are directly answered with a final reply stage (either a ResultStage, an ErrorStage or a CancelStage).

This is for example the case when you are signing with a seal on a server. No consent is required and the signature is applied.

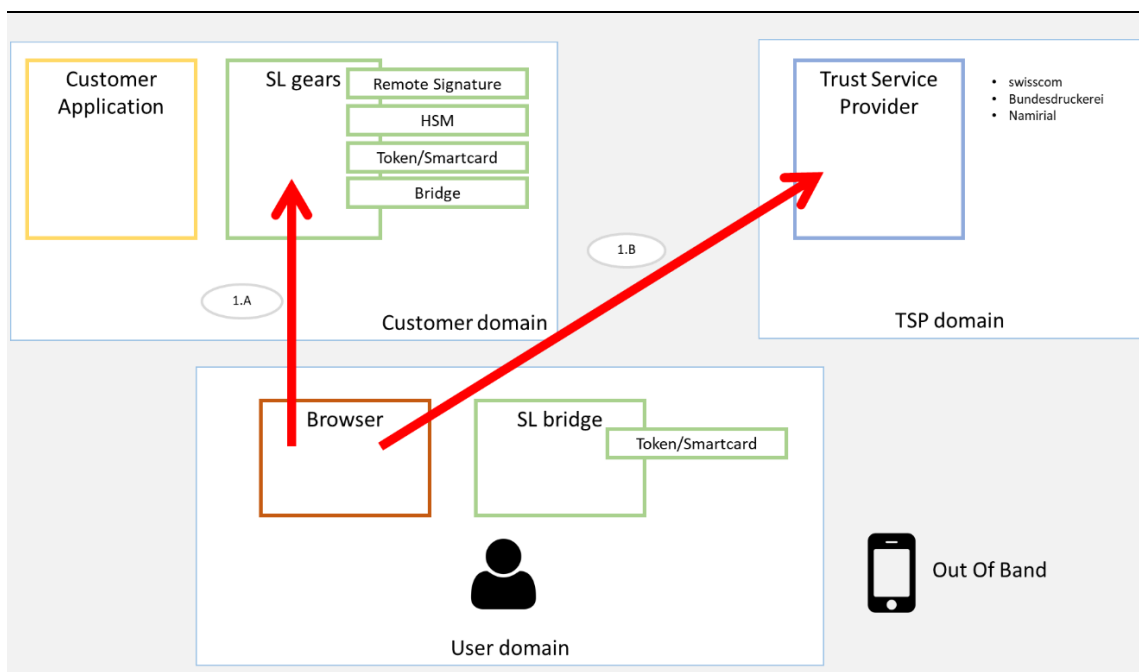
A process is initiated, here with a browser in your customer web application. Your application then calls gears. gears may issue more requests itself, for example to a TSP for signature creation, OCSF or CRL collection or whatever reason (this is important for the technical infrastructure later).

The outcome is communicated immediately back to your application as a ResultStage, ErrorStage or CancelStage.

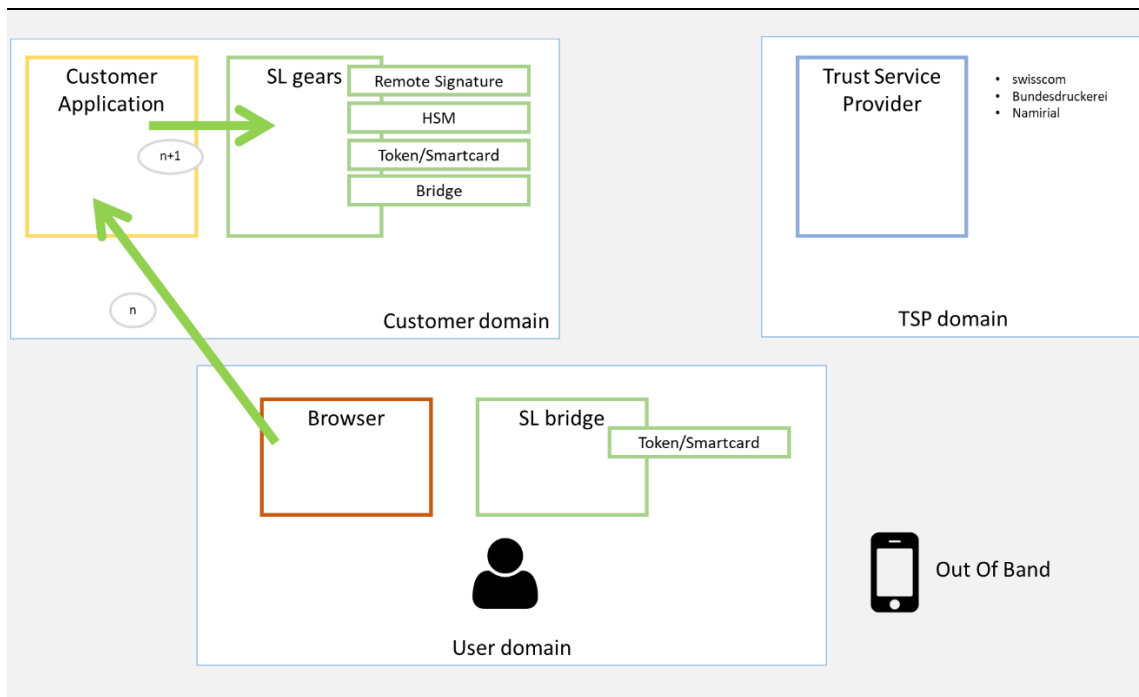
3.2 Asynchronous requests, front channel signalling

If the request cannot be fulfilled with a single request/response, you will get another ReplyStage, most probably a HttpResponseRedirectStage. This will indicate that some interaction steps have to be applied to reach the conversation goal.

The process itself is completely transparent to your application – you just have to "follow the orders" given in the ReplyStage, e.g. redirect the browser to a new target (the red arrows).



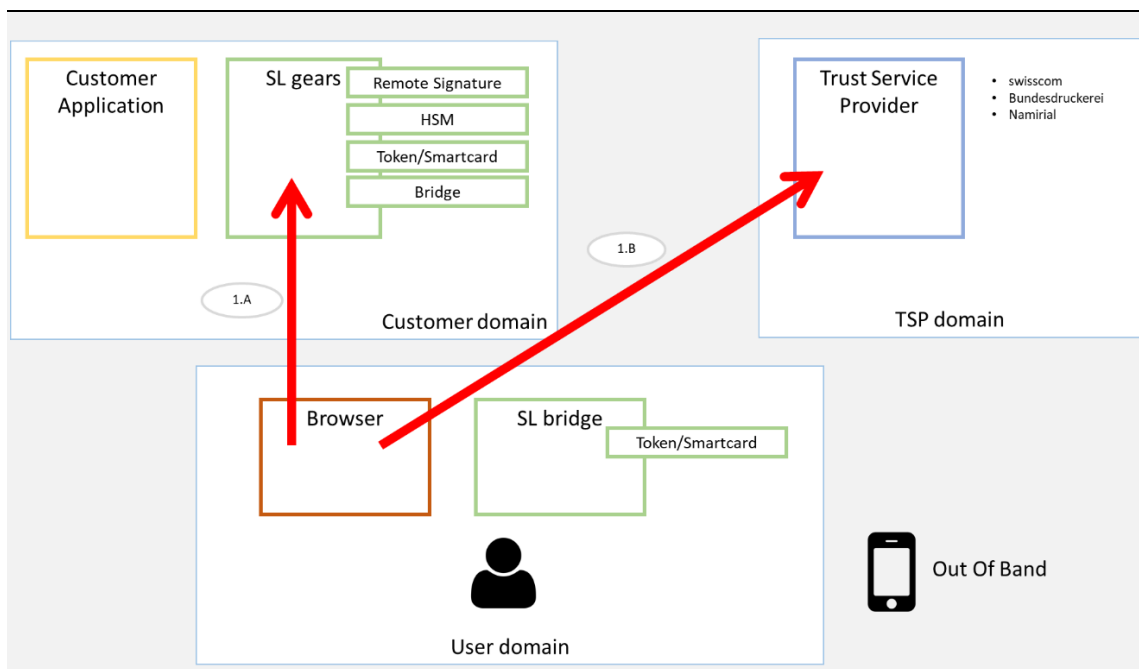
Eventually the conversation ends in some way and the process is redirected again to the callback address you have given upfront via a browser redirect.



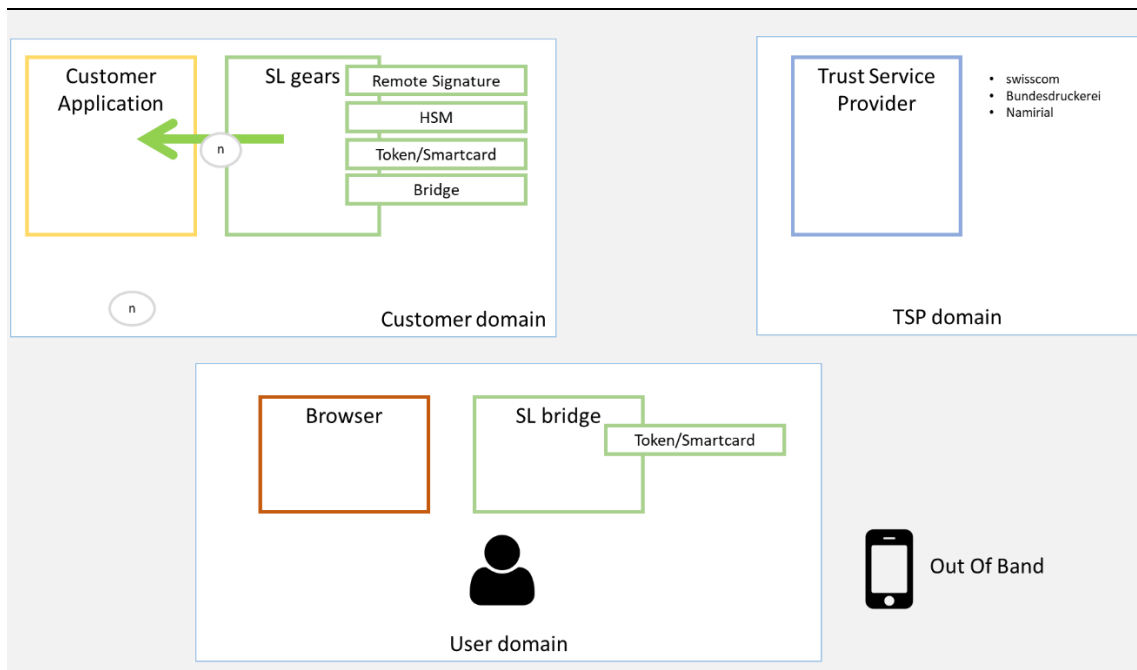
Your application contacts gears once more to collect the conversation outcome.

3.3 Asynchronous requests, back channel signalling

The other scenario is very similar until the conversation ends.



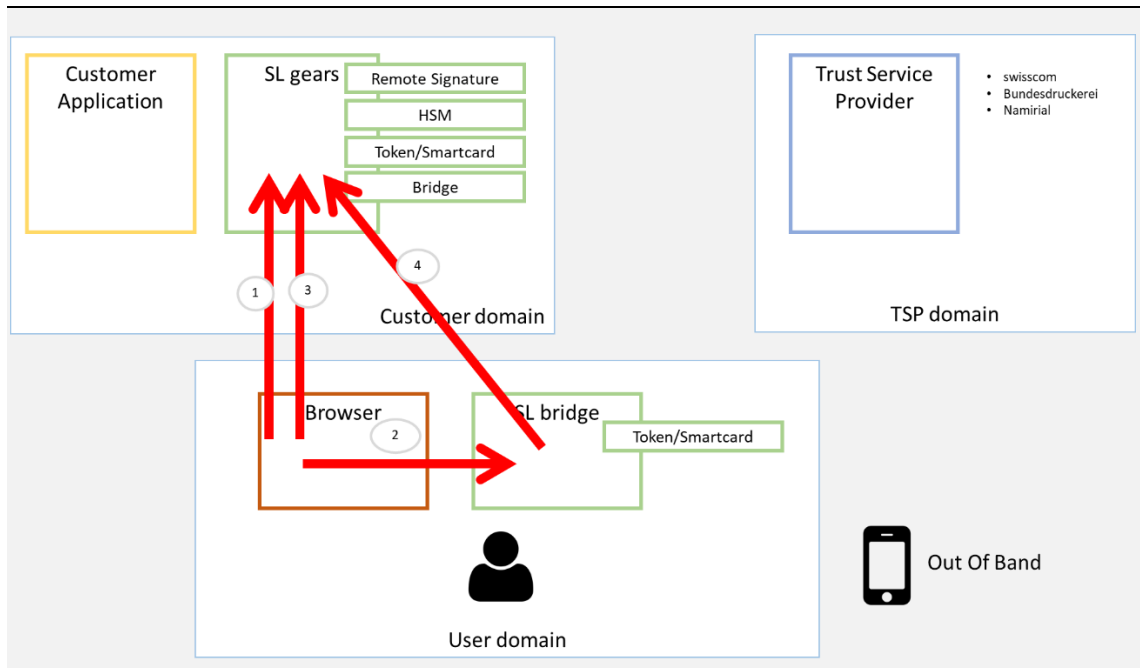
Now the conversation end is signaled via a backend request to a well-known address in your application.



3.4 Asynchronous requests with multiple participants

The conversation started by your application may initiate quite complex subprocesses you are not aware of.

E.g., some processes involve more than only the browser to fulfill the business goals. When you want to access the smartcard on your local computer, the bridge agent component will participate to bind the resource to the conversation and the browser.



Again, after the conversation ends, the signaling described in the chapters above take place.

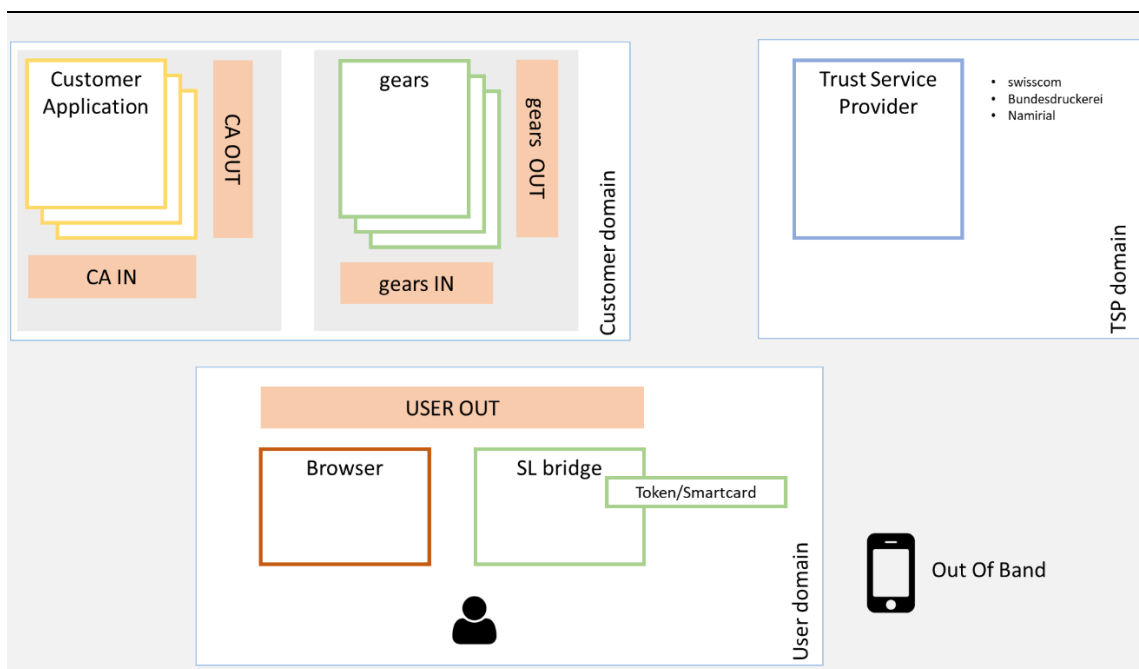
4. Network architecture

4.1 Overview

This chapter discusses **very** shortly network infrastructure and typical techniques and caveats.

The next picture is a sketch of the possible technical devices that may interfere with the communication paths outlined in the chapter before.

We are not interested in user domain or TSP domain, though.



The "worst case" architecture has a swarm of customer applications and a swarm of gears instances separated in two sub-domains.

Customer application sub-domain is shielded by networking devices like a reverse proxy or load balancer (CA IN) and a firewall (CA OUT). The customer application does normally not communicate with the TSP domain.

The gears sub-domain is shielded by another proxy / balancer (gears IN) and finally it has some firewall (gears OUT) to the TSP domain if a TSP is involved.

The user is guarded by devices in USER OUT.

4.2 Firewall

For sure your internal “CA OUT” must be configured to reach gears on its internal network. gears only provides HTTP(S) services on a single port.

“gears OUT” must be able to reach all required services from the TSP(s), e.g.

- Signature and authentication endpoints
- OCSP and CRL endpoints
- Timestamp service endpoints

The required ports and protocols depend on the TSP implementation.

In addition, it must be able to reach the customer application subdomain if back channel signaling is active.

4.3 Proxies

If you use a proxy / load balancer (gears IN) in front of your gears installation **and** you have asynchronous requests, you must be sure that gears is aware of its external address to properly generate redirect URLs.

By default, gears tries to derive its external address (in descending prio)

- from its static configuration (see [1])
- from the x-forward-* HTTP headers
- from the HTTP protocol information

4.4 Load balancing

4.4.1 Typical load balancer strategies

A load balancer tries to distribute all requests to a set of backend machines based on a given strategy (e.g. round-robin, least-used, ...).

All these strategies must deal with the concept of "session", where multiple requests are required to implement a business function. In this case the backend machine either itself shares the business state between the active machines **or** the load balancer has to take care to route all requests to the same machine (**sticky** behavior).

gears does not implement state sharing and so you have to implement **sticky** behavior on the load balancer level. The features provided by a load balancer may differ, but we present the most typical ones.

4.4.2 Hash based load balancing

Some load balancers provide "quasi static" routing by applying some hash function to request properties like “client IP”.

This can be applied even if the backend machine is completely unaware of being "load balanced", but it breaks in many scenarios that are quite common for gears.

E.g., a typical “viewer/create” request comes from your backend application and result in redirect for the browser. Both requests must be routed to the same backend.

4.4.3 Transparent load balancing

We call load balancing “transparent” when there is no intervention of the gears backend required. The load balancer adds some protocol level information like a cookie that carries information on the required gears backend machine.

Downstream participants like the browser or your application must be aware of this information and support propagation.

In the simplest scenario, where a browser directly interacts with a web application behind a load balancer a "sticky cookie" works beautifully.

4.4.4 Explicit load balancing

Load balancer and gears backend machine can be cooperating for sticky support. In this case the protocol element that gets inspected for determining the backend machine is created / maintained in the gears backend itself.

This allows fine grained control and more complex communication scenarios than browser <-> server.

All of the above scenarios are supported here

- Simple one-shot requests
- Multi step signatures
- Viewer (even multiple)
- Integration of non-browser agents (like the bridge)

5. gears

5.1 Conversations

Request processing in gears is coupled with the conversation concept, a conversation being a short-lived, session like object.

This conversation is not shared between gears instances so that each conversation related request must reach the respective gears instance – regardless from which client. Possible clients are the customer application, the browser or other participants (like the bridge agent).

A conversation does not necessarily require multiple requests, e.g., a non-consent signature request to a seal device may be performed in a single request / response cycle. This would not require resuming a conversation in another request.

5.2 Protocol specifics

gears itself identifies its conversation

- using the "conversation" property in a POST JSON request object for all official gears API requests
- using undisclosed parameters in internal protocols, both for POST and GET requests

These parameters are not intended to be used directly in load balancing strategies as some are subject to change without notice.

If gears **actively** takes part in a load balancing scenario, we build on a mechanism that is independent of the existing protocols and content.

5.3 Builtin load balancing support

Gears implements a simple mechanism that co-operates with an optional load balancer purely on the HTTP protocol level (URLs and headers).

There is no need for payload inspection.

The implementation can be activated using a spring profile. This will result in gears decorating each response *and* each URL it is emitting with an id that can be recognized by the load balancer.

This enables that each request that belongs to a “session” (i.e., a conversation) can be routed to the server that initiated it.

Details on the load balancing support are described in the manual.

5.4 Usage scenarios

5.4.1 Synchronous requests

Synchronous requests as detailed in “Reference architecture” do not require special setup in the incoming devices “CA IN” and “gears IN” as there is no follow up request that needs to be routed to the same instance.

5.4.2 Single instance gears

In case we only have a single gears instance, again we do not require any special setup.

5.4.3 Asynchronous requests

Things get more involved when asynchronous conversation processing starts and multiple gears instances are active.

In this scenario, the load balancing device must support **sticky sessions**, i.e. after the initial selection of a gears instance all subsequent requests for the same conversation must be routed to the same instance. An overview of some of the available techniques is given in the chapters above. The next chapters discuss their use in a gears architecture.

5.4.3.1 Hash based load balancing

Hash based techniques cannot be simply applied to asynchronous gears scenarios. If you revisit the communication paths in chapter “Reference architecture”, you will see that requests that act on the same conversation can stem from different physical machines (at least your client application and a browser) and do not share a common request format (POST/GET, parameter serialization).

5.4.3.2 Transparent load balancing

In this scenario, gears is not aware of a load balancing device. The load balancer typically adds a HTTP cookie.

All other HTTP headers will generally not work because the least intelligent client is assumed to be a browser – only sending cookies along with subsequent requests automatically.

Your client application must be aware of the cookie and store it for the duration of the conversation to be able to get all further “acknowledge” requests dispatched to the correct gears instance by adding the original cookie to the request.

In addition, your application is not the only component that needs the cookie for further communication. In case we need to redirect the browser (e.g., when forwarding a redirect to the gears viewer), your application needs to add the cookie in its redirect response. Remember, to make the browser accept this cookie, **the customer application and gears must share the same domain**. If this is the case, the browser will send this cookie automatically along in subsequent requests to the external gears address (gears IN), allowing the identification of the correct instance.

In a simple gears viewer or signer scenario, this will work.

For more complex scenarios with multiple participants (e.g., leveraging the bridge agent), this header would have to be forwarded between all participant to make all of them meet on the same server. This is not possible with "transparent" sticky sessions (without gears knowing of the load balancing mechanics).

To summarize:

- we rely only on standard HTTP features
- gears is not aware of the load balancer
- your application needs to take an active part in the communication (is aware of the load balancer)
- multiple participants do not work

5.4.3.3 Explicit load balancing

gears supports sticky load balancing by adding required information to protocol elements himself.

The goal is to relieve load balancer, client, browser and other participants from taking an active role, i.e., they are not required to add special headers or rewrite URLs.

The load balancer is only required to collect and inspect the sticky information elements and assign the appropriate machine.

Your client only needs to forward or use the URLs provided in the protocol.

The load balancing module in gears provides support to

- Instrument all internal redirects with an additional query parameter
- Add a HTTP header for special purposes
- Enumerate all conversation related URLs to be used by the client to drive the conversation

For more information on the load balancing configuration see the manual.

In this scenario you must for sure ensure that **each** request to gears goes through the "gears IN" instance.

5.4.4 Bridge

Using the bridge in a load balanced environment is possible only with the gears explicit load balancing module activated – but then it integrates in a well-defined load balancer and plays nicely in the conversation!

6. External References

- [1] intarsys GmbH, Sign Live! cloud suite gears manual.
- [2] Swisscom AG, "All-in Signing Service Reference Guide," [Online]. Available:
http://documents.swisscom.com/product/1000255-Digital_Signing_Service/Documents/Reference_Guide/Reference_Guide-All-in-Signing-Service-de.pdf.
- [3] intarsys GmbH, Sign Live! cloud suite gears cookbook.
- [4] intarsys GmbH, Sign Live! cloud suite gears incubator.
- [5] intarsys GmbH, Sign Live! cloud suite gears tutorial.
- [6] intarsys GmbH, Sign Live! Security Applications Developers Guide.