Developer guide:

**Introduction :**

Integration of different systems raises the need to provide a secure channel between those systems. For creating secure channel in the system, the application must to share a secret (in case of symmetric cryptography - private key. And in case of public cryptography- each application need to know the public key of the other, and must to Ensure the authenticity of this public key ) .

To solve the above problems we build key management system. In this system we allow the system administrator capability of - impose a task of creating keys on one program, and deliver this keys securely to other programs.

The system will be divided into two parts:

**Agent** - software that is installed on each client. This software will run continuously on the client computer. This program will get tasks from the system administrator, and will perform those tasks.

There are two kinds of tasks. Task that related to keys (like create new secret key – or install trust certificate in a keystore). And other kind of tasks that related to the agent management (like change the agent configuration ...).

Because some programs (that use our system) may run on one computer, and each program may use different type of keystore, the agent program will support plugins architecture, when each plugin will support different type of keystore. For each program that needs to use keystore, will be supported an instance of the suitable plugin , that will take care of this keystore.

All the tasks that related to keys will be done via the suitable implementor instance. So each implementor’s plugin need to support operation of -creating new keys, and store keys that deliver from the server.

This allows the system administrator, to ensure that two different programs share a common key and are able to communicate safely.

**Server** - provided a web site for the system admin. Via this site the admin can impose tasks on the agents, and also to follow the agents operations. The web-server also delivers the tasks and the keys to the agent. (Details and examples about the server UI operations found later in this document).

This guide gives a general explanation about the structure and the flow of the program. In order to help to those who want to continue and developing this program in the future. This guide contains of 3 sections, one about the agent , one about the server and one about the communicating protocol of this system .

Detailed explanation for each class can be found in the javadoc files that attached to the source code .

**The agent program :**

**Introduction :**

The Agent software where written in java and composed of two parts- the registration part and the service part.

In the registration part we establish secure connection channel between the agent and the server. And register the new agents in the server.

The service part is the main part of the agent program. In this part the agent runs in a infinity loop(until he get stop command) and act like that :

1. Asks the server for tasks .
2. Performs the tasks
3. Sleeps for X time (when X define in a configuration file).

In order to use the software logically, the agent must have one or more implementor instances. Each implementor instance has unique name.

There are two kinds of tasks that the agent can get.

One kind of tasks is: tasks for particular implementor. These tasks will be related to keys and after getting this kind of task - the agent delivers the command for the relevant implementor (the agent will recognize the implementor by his unique id ).

The other kind of tasks is general tasks for the agent. These tasks supposed to be tasks that not related to keys, like “create new implementor instance“ or “change configuration”. In this case the implementor Id that delivers in the message suppose to by “nop”.

**The registration – creating the First trust:**

This is a program that registers a new agent in the server, and creates secure channel between the agent and the server. (The secure channel base on ssl connection between the client and the server . The client has the server certificate in his the truststore and certificate that signed by the server in his key store).

If this program terminated successfully from it's run, the agent were installed on the server. Then the client can start to run the agent service program.

**Registration program Flow:**

1. Read all required parameters (from the configuration file or from user input) .

2. Change the agent name in the agent service configuration to be suitable to the agent name that now registering.

3. The agent creates new keystore and truststore in the agent service directory (these key store and truststore will be used to create a secure connection with the server).

4. The agent installs the certificate of the server in the trust store

5. The agent sends to the server request for registration. (The request sent over ssl channel without client authentication, the authentication will done by the installer name and password ).

The request contains the flowing details:

Csr - certificate request that the server need to sign.

Name – the id of the new implementor.

Implementers- the plugins that installed on this agent (found in the plugins dir of the agent service)

instName - the name of the installer (for authentication).

InstPassword - the password of the installer (for authentication).

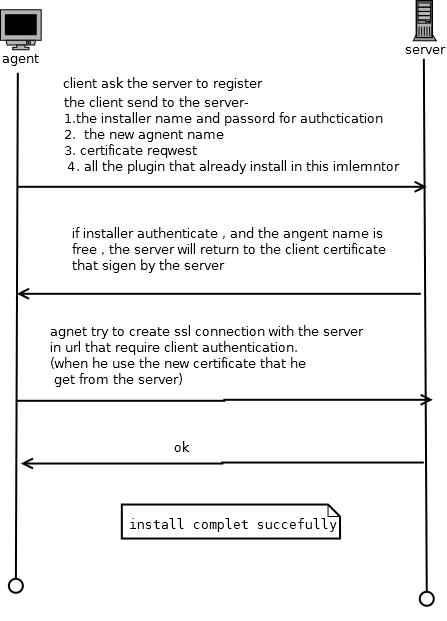
jsonConf - the agent configuration in json format

6. The server signs on the certificate and sends it back to the agent. The agent will install the new certificate in the keystore. If the username or password are wrong or the agent name is already taken the installation will fail .

7. agent tries to create secure connection to the server (on url with client authentication).

8. The server send confirmation message about the trust connection and the installation is completed .

Connection diagram :



**The Classes:**

In this section we will describe the java classes of creation of the first trust.

**class GenerateCSR**  **-** this is the main class of the program . Manages all the processes of creating of certificate request , and responsible for all registration processes.

**Class ArgsConf –** deals with the install parameters, reads the parameters that not supplied in the configuration file from the argv. If some parameters was not supplied in configuration file and not supplied in the argv, it asks the user for those parameters .

**Class Register-** responsible to step 5, in the flow that describe before.

**Class TrustConnectionCheck -** responsible to step 7, in the flow that describe before

**Class AgentIstallException –** an exception for the agent install process.

**Class MyKeytool –** class that helps to deal with keystore operations in java . (to create keys , and to save keys and certificates ) . All operations that performed in this class starts with loading the key store form file and ends in saving the keystore object back to the file. So there no problem of synchronization betweenthe file and the object.

More detailed explanations about each class are located in the javadoc of the project.

**The agent service project :**

After the installation, the user has to run the agent service. The agent service goal is to perform the tasks that it receives from the server.

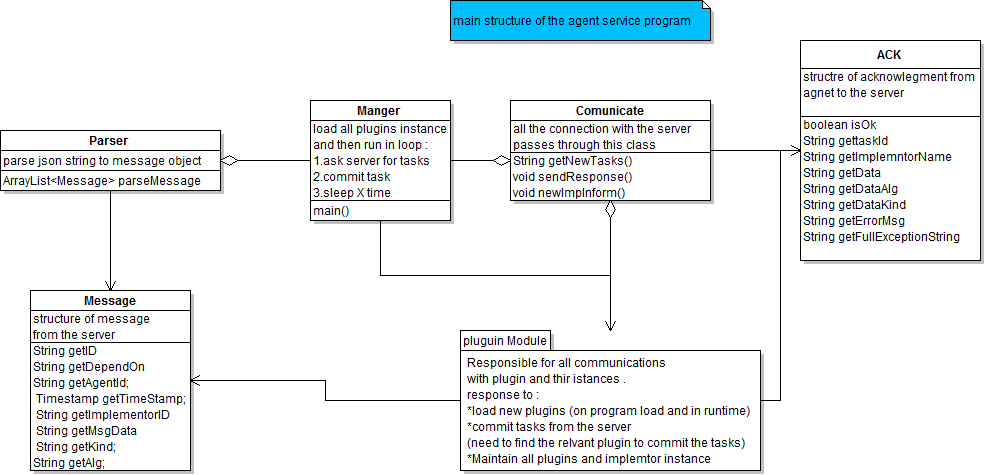
The configuration of the agent is located in the file “conf.cnf” in the agent service directory.

New plugins need to be located in the plugin directory. User can add new plugin also while the program is active, and the program will recognize the new plugin, and will load it dynamically.

For each implementor instance there is a property file in json format. Those file is located in the “inst” directory, When the ID of each instance is also the file name . Because those files may contain secret data (e.g. keystore password … ) the files are encrypted with the key file “CA.ico” that located in the “plugins” directory.

**Program structure:**

This is the UML diagram of the program:



**Manager class–** The class that contains the main function. Responsible for the scheduling of all modules in the program (this class contains the main loop of the program - step 4 in the program flow that describe above).

**Parser Class –** take json string that contains the tasks for the agent, and parse it into message object.

**Communicate class –** responsible to all communication with the server. Pull the tasks, send the ACK’s . And inform the server about new plugins.

**Message class –** contains the data of the message from the server to the client.

**ACK class –** contains data of acknowledgment about tasks sent from the agent to the server.

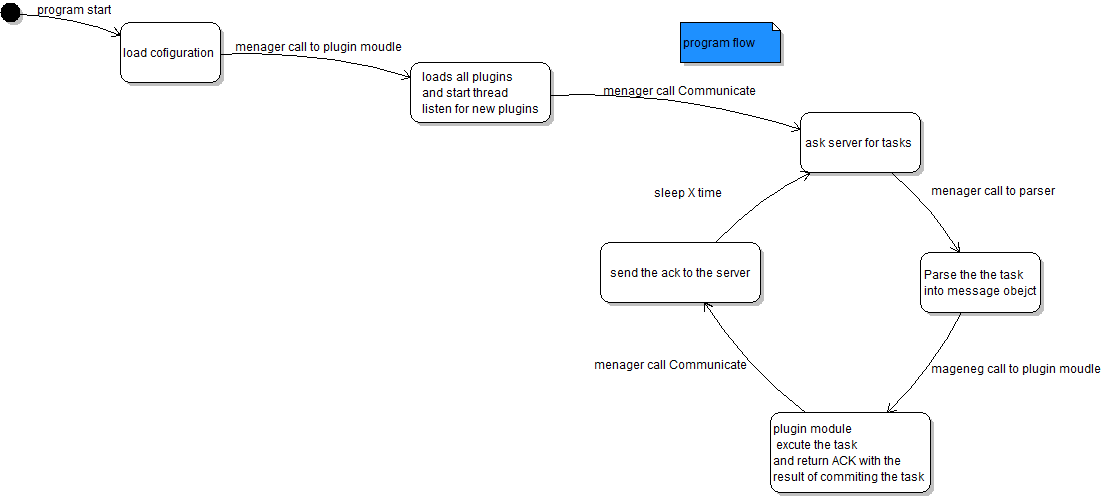
**Plugin module –** commits the tasks , and manage the implementers instances , and the plugins for this agent .

(More detailed explanations about each class are located in the javadoc of the project.)

**Program flow:**

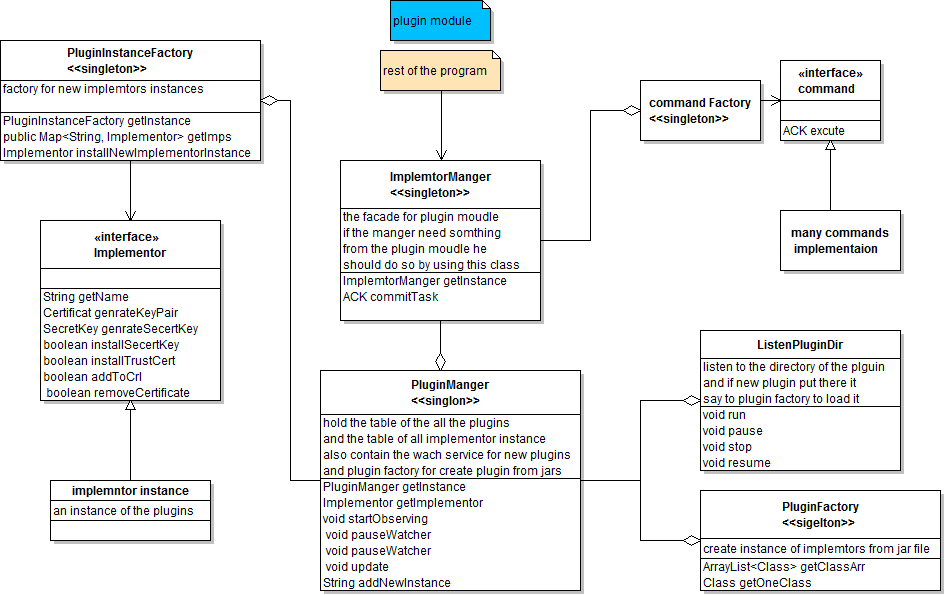
1. Agent loads the configuration from file.
2. Agent loads the plugins and the implementor instances that already installed.
3. Agent starts new thread that listens to the “plugins” directory.
4. While did not get a stop command:
   1. ask the server for tasks
   2. use the parser for create message’s object
   3. For each message that created by the parser:
      1. deliver it to implementer manger
      2. Implementer manger commits the task, and return ACK object.
      3. Send the ack back to the server.
   4. Sleep for X time (where x define in the configuration file).

**Program flow diagram :**

****

**Plugin moudle :**

The UML diagram of the plugin module:



**Implementer manager class –** façade for the plugin module. This in the only class that known to the rest of the program. Two main functions – commits tasks for committing tasks, and getInstance to load the plugin module.

**PluginMager class** - all the management of the plugins is done here. It contains two lists: one of plugins (abilities, java “Class” objects) and one of implementer’s instance. Contains also two factories for creating plugins and implementor instances.

**PluginFactory class –** has abilities of create new plugin class from jar file(only when the jar file contains an implementation of the Implementor interface).

**PluginIstanceFactory class-** has the abilities of create new implementor instance, from a plugin class and parameters to the contractor.

**Implementer interface –** an API for plugins that deals with keystores. Contains functions of creating saving and deleting keys.

**ListenPluginDir class-** runs in different thread, listens to the “plugins” directory. If new jar file added there, it calls to the plugin manager to load the new plugin and inform the server .

**Command interface** – an API for command. Gets implementer and message and perform the task .

**CommandFactory class –** gets a message and return suitable command instance

**Plugins module scenario’s:**

**commit task in plugin module:**

1. Implementor manager get message
2. Implementor manager get the suitable implementor instance from implementor manager
3. Implementor manager get the relevant command from the command factory
4. Implementor manager execute the command
5. Implementor manager return ACK to the server

**dynamic loading of plugin(while program running) :**

1. ListenPluginDir find new jar file in plugin directory
2. ListenPluginDir deliver this massage to pluginManager
3. PluginMager use the plugin factory to add this plugin
4. PluginMager inform the server about the new plugin

**Static loading of plugins(while program loading):**

1. Implementor manager loaded (with get instance command)
2. Plugin Factory loads all the plugins in the “plugins” directory
3. Plugin manager informs the server about all the loaded plugin .

**Commands –**

**Commands for implementer :**

*generate key Pair* - generate new key pair in return the public key in x.509 certificate

*generate secret* - generate new secret key , and deliver it to the server .

*install cert* - install this certificate in the trust store of the implementer

*install secret* - save this secret key .

*remove certificate* – command to implementer – remove this certificate from trust store

*add to crl* - command for implementer – add serial number to certificate revocation list file .

**command for agent :**

(for this kind of command the implementer name in the message is “nop” )

*new inst* - command for agent – add new implementer instance of plugin

*change conf* – command for agent – change the agent configuration

**The server program :**

The server program written in php .

All the configurations of the server stored in the configuration file “conf.cnf” .

The program needs to run under apache http server that enables php module and sll module .

The server program has two main functions:

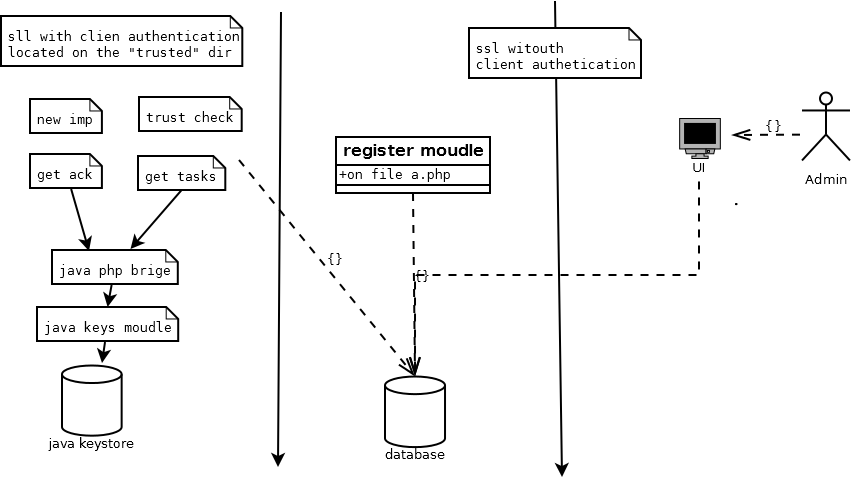
1. To provide a web site for the system admin. From this site the admin can create new tasks for the agent and their implementers. And it also monitors the actions of agents.
2. To deliver the task to agents where they ask for.

All the connections with client needs to be above ssl with server authentication, while the connection with the agents (except the registration part) needs to contain also client authentication . (the client authentication in the UI part and in the registration part done by password) .

All the data in the sever saved in the database. Except the case when it needs to keep key. That because we want to save the keys in safe key store, and php not supports appropriate key store. So we use the “php java bridge” project, and connecting to a java module that store the keys in java key store.

All the tasks and the data about the tasks saved in the database. If agent pulls the same task X times the mission is aborted. It also removed from the table of the tasks, and added into the failed tasks table. In contrast if the agent sends ACK about tasks that performed, the task will be delivered to the done tasks table.

**Server structure :**

****

**The agent communication module:**

In the left side of this diagram we can see the section with the client authentication .

The modules in the left side :

Trust check : check that registration was successful.

New imp: the agent informs the server about new plugin.

Get ack – when the agent returns ack to the server. This due to the fact that sometimes the ACK contains key and we need to save this key. This module is connected to the java php bridge for save the keys in the java key store .

Get tasks – when the agent asks the server for new tasks. This module connects to the “java php bridge” because sometimes we need to deliver key or certificate with the task, and we need to pull those keys from the java key store.

**The registration module :**

In the middle of the diagram we can see the module of registering the agent.

Using this model occurs when it is new agent that wants to register .

The agent installer authenticate himself by username and password .

The agent asks for agent id – if this id isn’t free the installation failed

For creating trust connection in the future, the agent sends certificate request to the server and the server signs the request (it uses in the php openssl library that uses the openssl program, the server uses key file and certificate file from the configuration file : “keyFiles.cnf”).

After the agent installs the certificate he can create a trust connection with the server. In the registration processes the agent will check that everything works while he tries to connect the trust connection check module.

**The UI:**

The UI were written in html , javascript, jquery and php.

For the design of the page we used the bootstrap library.

The UI have 2 basic operations :

1. Show data about the agents and their activity
2. Generate new task for the agents .

When user wants to use the site . He needs to authenticate himself by username and password . After this the new session is opened for this user in the server . If none action is done in the site in half an hour, the user will be disconnected and needs to login again (this checks happens in the function “chekSession()” in the php module "chekSession.php").

**The data base:**

**Tables :**

Agents - contains the agent last connection data , and his register date

Agentsconf- contains the current configuration of each agent.

Algorithms- contains all the secret key algorithms that the implementer supports

Donetasks – contains tasks that done successfully.

Failedtasks – contains task that failed (the agent pulled them more then X time but did not commit them)

Implementors- table contains the agent name and id of implementer instance

Inreg – agent that in the middle of the registration processes. The connection was created to the register module but still not connected to the check trust connection

Lowsecuredata – table that contains data that related to task, but the data is not a key or certificate

Permission - table that contains usernames and hash on password for check user for login and for installing new agent (the hash on the password is sha-256 when we add random number to the password , then do the hash and then add the random number in the start of hashed word for increase the security ) .

Plugins- tables that contains all the plugins for each implementers

Serverlog- log for all the operation in the server

Tasks tasks that wait for agent to pull and commit them .

**Communication:**

**Introduction:**

All the communication between the client and the server done above ssl socket. Also accessing the php files that located in the “trusted” directory requires client authentication (in the ssl handshake).

In the server we use apache http server with mod\_ssl, inside the apache httpd\_ssl configuration file we need to declare that the access to our project requires ssl. We also declare in this file that accessing to the trusted directory requires also client authentication.

For doing this declaration you need to add into the apache httpd\_ssl configuration file those lines: (it done during the installation automaticly in windows , and manually on linux ).

<Location /{yourProjectDir(inside the htdocs})>

SSLRequireSSL

</Location>

<Location /{yourProjectDir(inside the htdocs})>/trusted>

SSLRequireSSL

SSLVerifyClient require

SSLVerifyDepth 10

</Location>

In the agent we use apache http client for creating the ssl connection , the keystore and the truststore for creating this connection located in the “keystore” directory inside the agent service directory . Those keystore and truststore are created in the in installation of the agent. Before the first trust created ,the agent have only the server certificate in his truststore, and after the trust connection established the agent adds his certificate (that signed by the server) to the keystore .

The communication with the server is always been initialized by the client and never vice versa.

**The tasks flow :**

The main part of this system is the tasks. The agent asks the server periodically for tasks . The server sends the tasks to the agent . After the agent tries commit the task he sends an acknowledgment to the server .

When the agent asks for task he sends his name as post parameter to the server .

The server return him array of task :

Each task contains those parameters:

task Id- number the id of the tasks

dependOn – if this task needs data from other task it will be the task id else it will be 0

alg – the algorithm needed for this task (If needed algorithm otherwise it will be “”).

kind – the kind of the task

impelmntorID – the implemtor that need to commit this task.

command date- the date this command was generated

Data- the data that related to the task .

This is an example of message for implementor that contains two tasks :

*[{"taskId":"12","dependOn":"0","alg":"X.509","kind":"generate key Pair","implementorId":"yakovApache","commandDate":"2012-08-16 13:10:12"},{"taskId":"14","dependOn":"0","alg":"","kind":"new inst","implementorId":"nop","commandDate":"2012-08-16 13:10:28","data":"{\"pluginName\":\"JksImplemntor\",\"params\":{\"ksPassword\":\"a10097\",\"ksPath\":\"C:\/temp\/keyStore\/my.keyStore\",\"ksType\":\"JCEKS\",\"tsPassword\":\"a10097\",\"tsPath\":\"C:\/temp\/keyStore\/my.keyStore\",\"tsType\":\"JCEKS\",\"name\":\"yosiJavaProgram\",\"algs\":\"AES,3DES\"}}"}]*

After the agent performs the a task he sends an acknowledgment about the task .

The acknowledgment parameters contains:

isOk- (Boolean) true on success and false on fail

taskId- the id of the task

impId- the imlementor that commits the task

agentId- the agent name

if the message has data so those parameters added to acknowledgement :

data- the data (if it is byte data encode in base64 ).

datakind- the kind of this data (e.g. certificate ,secret key …).

dataAlg – the algorithm of this data (e.g. x.509 , AES … ).

if the message is error message so those parameters added to acknowledgement:

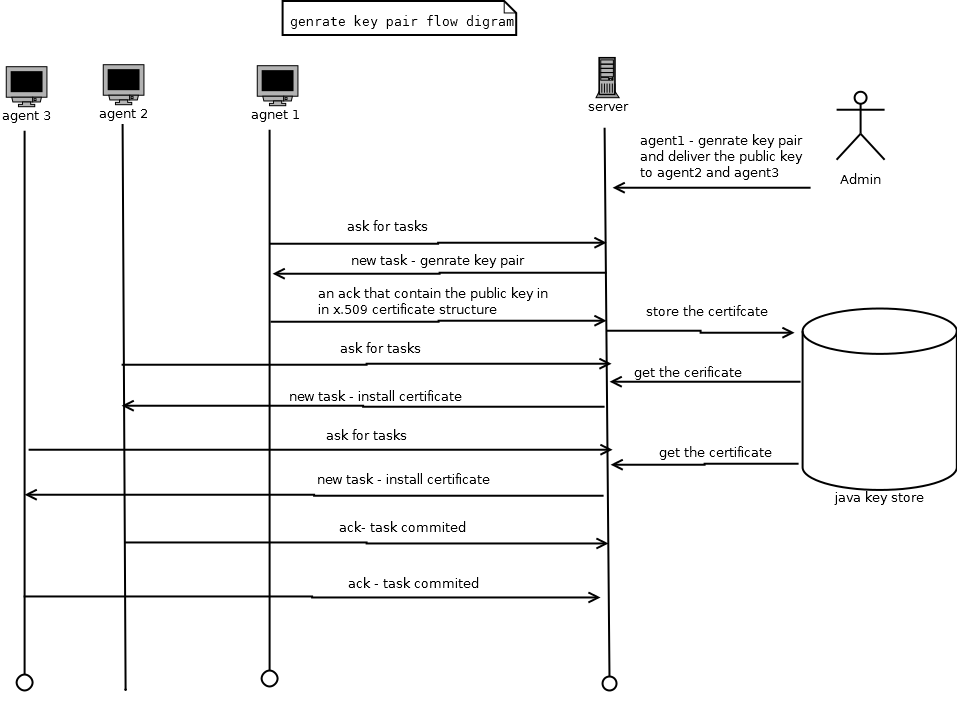
errorMsg- short explanation about the error

fullExecption- full stack trace of the exception in eclipse

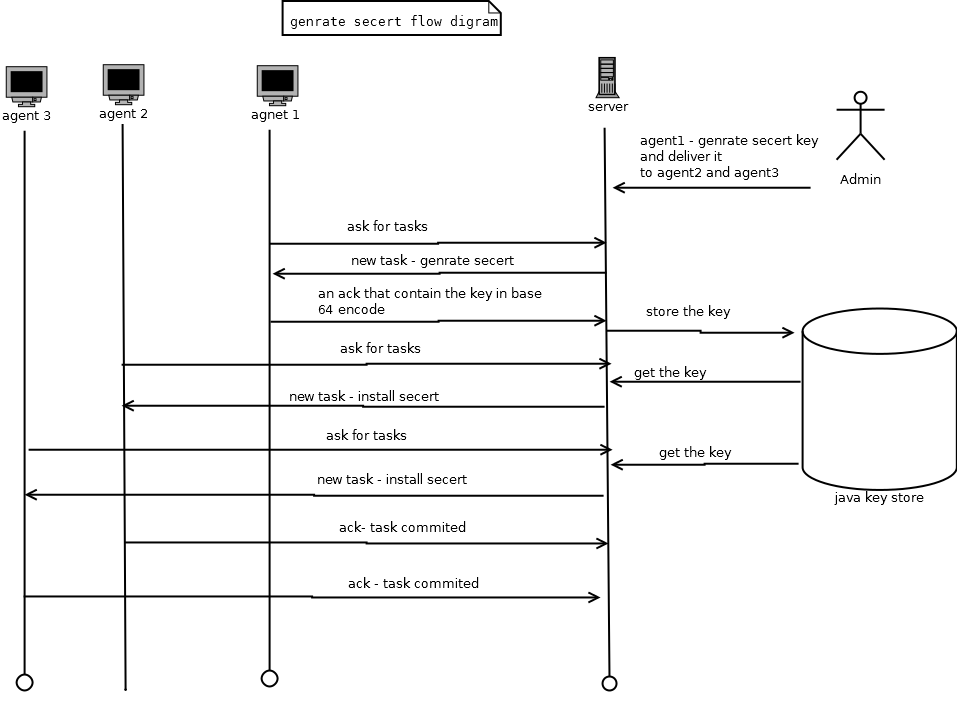
The acknowledgment send via post – and all those parameters delivered as post parameters .

**Tasks flow diagram :**

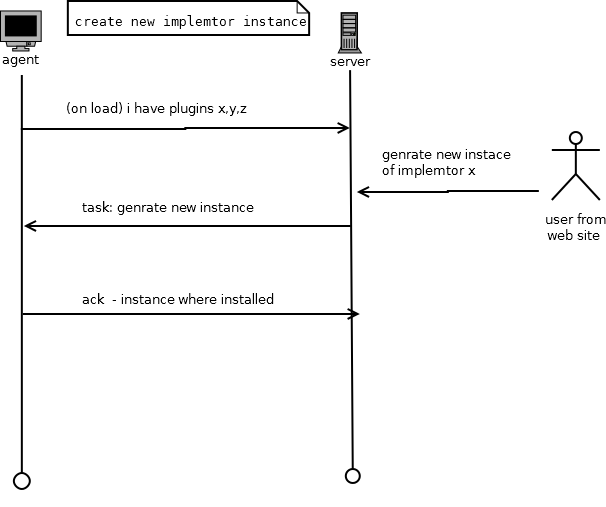
Generate key pair :



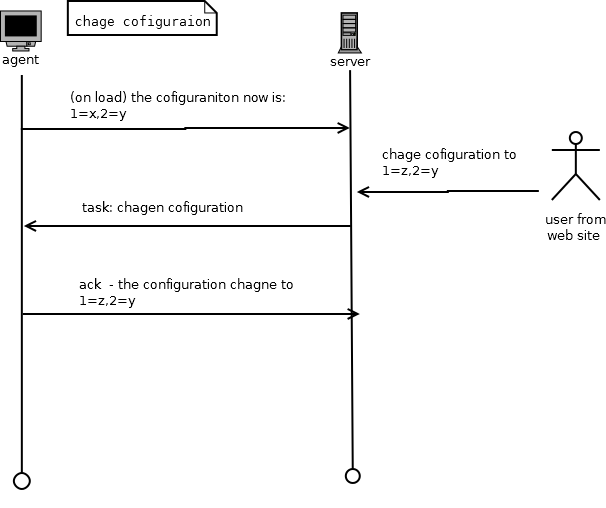
Generate secret key:



New instance diagram :



Change configuration diagram:



Case of error :

