Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

ViSaG - Virtual Safe GRID

POP-C++ Virtual-Secure (VS)

Road to ViSaG

Author:
Valentin Clément

Date: April 5, 2011
Revision: 1.5

Contents

1 Introduction

Virtual - POP-C++ is a version of the POP-C++ middle-ware that would be able to run parallel objects into virtual machines. The development of this version is the core of the ViSaG project. A prototype version of Virtual - POP-C++ has been developed during a bachelor thesis project in the summer 2010. This version is called VirtualPOPC-1[1].

NOTE : VirtualPOPC-1 will be referred as VPOP1 in the rest of this document.

This document is structured as follows:

- The next chapter is focusing on the weaknesses of the prototype developed during a Bachelor Thesis.
- The Chapter 3 aims to dispatch the work between the partner institutions working on this project.
- The Chapter 4 was written to helps the project team to take the right decisions concerning the future development on this project.
- Chapter 5 explains all the modifications done on the first prototype to make the final version of POP-C++ for ViSaG.
- Chapter 6 just explains small modifications that are general to all POP-C++ versions.
- Chapter 7 tells what we think about the security of our new version.
- Chapter 8 focuses on functional and performance tests.
- Chapter 9 gives some ideas for future development.

Keep in mind this document has been updated during the whole development process. The firsts chapters have been written before the actual development of the new versions.

2 Requirements for Virtual POP-C++

This chapter aims to identify the weaknesses and the missing elements of VPOP1. At the end of this chapter, all the modifications that have to be brought for the final version to be produced for the ViSaG project will be summarized and shared between the EIA-FR and the HEPIA.

2.1 Hypervisor related data

In VPOP1, all the data related to the hypervisor connection are saved as environment variables. These data should be saved in the JobMgr config file located at *POPC_LOCATION*/etc/jobmgr.conf. The hypervisor password should be encrypted and not saved in clear.

Files to be modified : ./lib/jobmgr.cc, ./include/jobmgr.ph, ./script/popc_setup.in

Work done:
All the hypervisor related data are now stored in a file located in the POP_LOCATION/etc/virtual.conf. These data could evolve with the changes that will be done in the future.

Decision:
The whole configuration file will be encrypted with a symmetric algorithm. More information about this point are given later in this document.

2.2 Code injection in wrapper

In the wrapper used to contact the hypervisor, some methods execute a shell command. As the command executed is prepared in the method, some malicious codes could be injected. To avoid this code injection, the commands should be controlled before their execution.

Files to be modified : ./lib/ESXWrapper.cc

Work done:
The wrapper does not use external command any more.

Decision:
The methods using shell command will be replaced by VIX API code. The cloning process will run some external command but this process is not executed in the wrapper. We are currently looking to implement the cloning function inside libvirt itself.

2.3 POP Application Identifier

A POP Application Identifier must be created for any POP-C++ application. This ID must be sent with the resource discovery request. A virtual node will create only one virtual machine (VM) per application. Any parallel object of the same application running on the same virtual node will be executed in the same VM. This VM will be shutdown at the end of the application or after a specified idle time.

File to be modified : ./lib/request.cc, ./include/request.h, ./lib/jobmgr.cc, ./lib/appservice.ph, ./lib/appservice.cc, ./lib/popc_search_node.cc

Work done:
The POPAppId is passed in the resource discovery process and the admin-VM takes it into consideration for the workers management. The VM is shutdown when it has no more jobs running on it. This step is finished.

Decision:
The VM could be shutdown when the application is finished. A message must be send to all nodes to shutdown the VM associated with the application. This decision has been implemented in POP-C++. More information about this implementation can be found later in this document.

2.4 Installation of admin and worker VM

The installation of the environment on the admin and worker VM could be the source of many problems. This installation should be as automatic as possible (missing library, ssh key generation, ssh_config, cloning VM ...).

Files to be modified : ./script/popc_setup.in, ./configure.ac

Work done:
The configure script is no able to accept the option "–enable-virtual". This option allows the user to compile POP-C++ in the Virtual version. Without this option, the standard version is compiled. To be able to compile two different versions, a VirtualJobMgr and a VirtualPOPCSearchNode have been implemented. All the virtual version is implemented by overwriting basic function of the standard JobMgr and POPCSearchNode. This step is almost done. There is some work to do in the installation script (remove useless questions, check SSH, generate key if not present ...).

Decision:
The popc_setup script as soon as we have the time.

2.5 Cloning VM

The worker VM must be cloned as much as needed on a virtual node. This process should be done before needing the VM. As the cloning process takes some time, it could be a good idea to start the cloning process when just one VM is available.
For example, a node running Virtual POP-C++ is using 3 workers. When the second worker is reserved (just one more worker free), the cloning process starts to make a new worker. The maximum number of worker on a node must be defined during the installation.

Files to be modified: Not defined

Work done:
Some experiments with VIX and with a home made script. Nothing implemented in POP-C++ for the moment. HEPIA has maybe some hints to give about this process.

Decision:
Using the hard copy solution from a C++ code. Asking HEPIA if they could find a better solution. The cloning process starts when there are only X free VM left. The number of VM prepared in advance is specified as an option.

2.6 Preparation of the worker VM

In the current version, the preparation of the VM is not very reliable. In fact, during this preparation, the ESXWrapper will try to get the VM IP. After three attempts, the preparation will fail if the IP is not discovered. This process should be optimized and maybe done differently.

Files to be modified : ./lib/ESXWrapper.cc

Work done:
Solved the waiting problem. Try with the VIX API, which seems to be more reliable than the command line tool. Nothing is implemented with the VIX API in POP-C++ for the moment but I got a test program sample.

Decision
Using the VIX API to retrieve the IP address. DHCP renewal must be done before retrieving the address (this point is open because more information on DHCP must be found).

2.7 JobMgr - Management of VM and jobs

As the JobMgr running on the admin-VM must be able to manage several worker-VM, this JobMgr must be modified to include the management of different VM with the POPAppID. This management must include the startup, check during runtime and shutdown of a VM for an application.

Files to be modified : ./lib/jobmgr.cc, ./lib/ESXWrapper.cc

Work done:
Modify the check and reserve process for taking into consideration the management of several workers. Ability to run several objects of the same application on the same VM. Host resource management must be defined.

Decision:
The admin-VM manage all the resource on the node. The all node is seen as one POP-C++ node. The VMs are just a way to encapsulate the parallel objects execution.

2.8 WorkerDeamon

The WorkerDeamon is in charge of the key exchange between the admin-VM and the worker-VM. This worker is not well developed. Here are some hints to find a different way to exchange the keys:

Hints:

- The keys could be passed during the installation of the environment. Once one worker-VM is cloned, its virtual disk could be mounted on the admin-VM and the admin-VM key could be write in the file system.
- The keys could be passed during the installation of the environment. Once one worker-VM is cloned, it could be started and the keys exchanged before taking a snapshot of it.
- A POP-C++ parallel object could be used to do that. The creation of a virtual POP-C++ service as a parallel object could run independently on the worker-VM. This parallel object could be contacted by the admin-VM and the keys could be exchanged.

Work done:
Test program with the VIX API and the copy file function. This solution seems to work perfectly. With this solution, the POP-C++ Security Manager running on the admin-VM could manage the keys of all workers. Nothing has to run on the worker.

Decision:
Use the VIX API to do this task. Ask HEPIA is they could find another way to communicate between the admin-VM and the worker-VM. We are aware that this solution is less generic.
Some other solution must be explored such as shared disk or shared network (libvirt).

2.9 Pseudo-Main

As we want to run a POP-C++ application in a full secured environment, the main of the application should be launched in a VM too. To do that, a pseudo-main could redirect the real main on a VM and launch it in a secure way.

Files to be modified : POP-C++ compiler (paroc_main)

Work done:
Nothing for the moment.

Decision:
This point is open. We can offer an option to let the user choose to run the main in a VM or not.

2.10 Fusion of POP-C++ 1.3.1 beta JS1.2 and Virtual-POP-C++

The Virtual version of POP-C++ must be merged with the secure version of the POP-C++. This will make the ViSaG version of POP-C++.

Work done:
Prepare the configure script to be able to accept the option "–enable-secure".

2.11 Minimal Linux distribution

The admin-VM should be as light as possible. In fact, this admin-VM will never run a parallel object for a POP-C++ application. This VM is only in charge of the management of the others VM known as worker-VM. Due to this fact, this VM should run the lightest distribution of Linux that can run the POP-C++ Global Services. This distribution should also support "libvirt" and the "vSphere CLI" libraries.

Work done:
Nothing for the moment. HEPIA could work in this.

2.12 Add restrictions on SSH

It's possible to add restrictions on the use of SSH for a specific public key. It could be a good idea to allow only what we really need for POP-C++ (SSH tunnelling and the popcrun command). This restriction are added with the public key in the "authorized_keys" file. The POP-C++ Security Manager should be able to add these restrictions when it writes a new key.

We can imagine that all SSH restrictions are defined during the installation of POP-C++. These information will be passed to the PSM for future usage.

Work done:
Nothing for the moment.

Decision:
Add SSH restrictions. Only enable the tunnelling and the use of the "popcobjrun" command.

2.13 ESXi user and restriction

VMware ESXi has a user and group management system. It would be more secure to create a specific user for Virtual POP-C++ and allow only the specific action related to its usage.

The administrator of the ESXi platform has many privileges that can be granted or revoked. All the actions about users and groups must be done from the vSphere Client.

Work done:
Nothing for the moment.

Decision
Using the root account for the moment. When the full version of POP-C++ Virtual and Secure will be ready, ask HEPIA to do some test if we can reduce the rights and use a different account.

2.14 Testing

A full protocol of test must be done on the full version of POP-C++ for ViSaG. The tests must contain :

1. Stress test to check the concurrency problems.
2. Performance test to be able to measure the overhead and make some optimizations.
3. Memory test using mudflap or valgrind to check any memory leaks.

2.15 Encrypted file system

The possibility to use an encrypted file system on the VM could enforce the security of the whole project. This point is open and must be studied.

3 Repartition

This section suggests a repartition of the work between the EIA-FR and the HEPIA (the two schools working on the ViSaG project and more specifically on the lower level of this project).

Theoretical, the EIA-FR is in charge of the major modifications of the POP-C++ middle-ware and the HEPIA is in charge of the global VM management. The task above will be

Section	What ?	Who ?	Priority
2.1	Hypervisor informations	EIA-FR	Medium (Done, except password crypt)
2.2	Code injections in wrapper	EIA-FR	Medium
2.3	POP Application Identifier	EIA-FR	(DONE!)
2.4	Installation of admin and worker VM	EIA-FR	(DONE!)
2.5	Cloning VM	HEPIA (EIA-FR)	On the way
2.6	Preparation of worker VM	EIA-FR	(Will be done with VIX)
2.7	JobMgr - management	EIA-FR	(on the way)
2.8	WorkerDeamon	EIA-FR	High (VIX API)
2.9	PseudoMain	EIA-FR	Medium
2.10	Fusion	EIA-FR	High
2.11	Minimal Linux distribution	HEPIA	Medium
2.12	Add restrictions to SSH	EIA-FR	High
2.13	ESXi user and restrictions	HEPIA	Medium
2.14	Testing	HEPIA and EIA-FR	High (wait full version)
2.15	Encrypted file system	HEPIA and EIA-FR	Medium

4 Open points and information

This chapter regroups some points that are still open and some useful information for taking decisions about the future step to do in the ViSaG project. Some of those points have been discussed and a decision has been made (see Chapter 2 for the decisions made).

4.1 VMware VIX API

VMware provides a C based API to interact with VM. This API has some interesting function that could help us in the development of the Virtual version of POP-C++. These functions are:

- **VixVM_Clone** : cloning an existent VM with snapshot (unimplemented with ESXi).
- **VixVM_CopyFileFromGuestToHost** : copy a file from the guest VM (worker) to the host VM (admin)
- **VixVM_CopyFileFromHostToGuest** : copy a file from the host VM (admin) to the guest VM (worker)
- **VixVM_ReadVariable** : some property of the VM (ip address) could be read like that.

The VIX 1.10 API reference can be found at following URL:
http://www.vmware.com/support/developer/vix-api/vix110_reference/

NOTE: The "product support" section of the reference cited above mentions that the function VixVM_Clone is only supported on VMware Workstation platform.

REMARK: Using a proprietary API could be a little bit restrictive but it would also be much more reliable than the use of command line tools that are also proprietary.

4.2 Cloning

The cloning process of a VM is going like this :

1. Create a new folder on the data store (vifs –mkdir).
2. Copy every files from the original VM folder to the new folder (except log files) (vmkfstools -i).
3. Modify the "display name" of the VM (vifs -g, awk, vifs -p).
4. Register the new VM in the inventory (vmware-cmd).
5. Start the VM (vmware-cmd).

This solution could be easily implemented in the current version of Virtual POP-C++ (a shell script is already working for that purpose).

NOTE(1): None of the VIX API or LIBVIRT is able to provide this function for the moment.

NOTE(2): When a worker is cloned, the SSH pairs is the same on the original VM and on the cloned VM. This is not a problem for the functionality of POP-C++. It could be a bad practice from a security point of view. We could force the worker to generate new key after the cloning process by launching a command with VIX.

4.3 Key exchange with the VM

The workerDeamon must be changed because now it is not reliable and not secure enough. In addition, with the secure version, the worker VM must be able to do more than just receive a key. Here are the needs for the key exchange:

- The admin must be able to read the PKI of the worker.
- The worker must accept PKI from the admin.

The VIX API gives a way to copy a file from a host to a guest. This API also gives a way to execute a program on the guest.

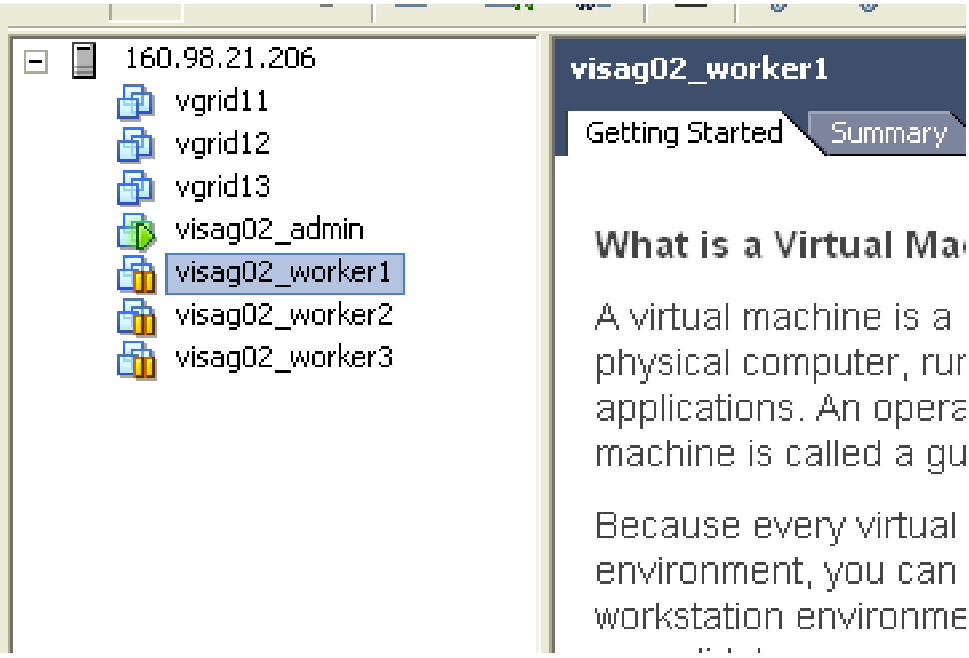
4.4 Names and addresses used in Virtual POP-C++

POP-C++ and its virtual version use different mechanism to retrieve a machine identity. Here is a brief summary of methods used in the virtual version of POP-C++ currently in development.

Between Global Services
Between the global services, either the IP address or the hostname could be used.

Between VM-admin and VM-worker
Between the VM-admin and the VM-worker two different kind of identification are used. Firstly, when the admin reverts the worker to the correct snapshot, the "display name" of the VM is used. This "display name" is a notion from VMware ESXi. A VM is registered under this name in the vSphere inventory (see Figure 1). With this "display name", the admin is able to retrieve the IP address of the worker. Once the admin knows the worker IP address it can communicate directly with the worker (at this point the hostname could also be used).

Figure 1: vSphere Client - inventory (display name)



4.5 DHCP

Using the DHCP service for the worker is not without any risks. This section will list the known issues about its usage.

Lease on a suspended VM
DHCP service uses lease for IP address. When a VM is reverted to a snapshot from a "shutdown state" or a "suspended state", the lease could be expired and the IP address may change after a few minutes. This behaviour could make the POP-C++ application crash. A solution to this problem is to force the VM to ask for a new lease when it is reverted. In this case the IP address would be changed before any execution on the worker and the application could run normally.

5 Modifications related to ViSaG

This chapter is a review of the modifications made on the prototype (VPOP1) to make a fully functional version of POP-C++ VS for the ViSaG project. The new version includes also the modifications made on POP-C++ for the Secure version (for more information, please refer to "POP-C++ over SSH Tunnel"[2]).

5.1 Compilation of four different versions of POP-C++

As POP-C++ could be installed for different usage (cluster, secure, virtual ...), the ability to compile the right version is fundamental. There are four different ways to compile POP-C++ by passing option to the configure script. Here are the four versions :

- **POP-C++** Standard version (cluster, local test...) : configure without options
- **POP-C++ S**ecure version (with SSH tunnelling) : configure with the option –enable-secure
- **POP-C++ V**irtual version (with worker virtualization - Admin Node only) : configure with the option –enable-virtual
- **POP-C++ V**irtual-**S**ecure version (SSH tunnelling and worker virtualization - for Admin Node only) : configure with the options –enable-virtual –enable-secure

NOTE: Be aware that all the versions are not fully compatible with each other. For the moment, standard and virtual version can be used together and secure and virtual secure versions can be used together. A compatibility table is available in the document "POP-C++ Virtual-Secure : POP-C++ User and Installation manual add-on"[3].

To make this compilation possible, the POP-C++ Global Services has been specialized with new classes. Figure 2 shows the new class diagram of the POP-C++ Global Services in each version.

All the modifications related to the VS version are now made in the VirtSecureJobMgr, VirtualJobMgr, VirtSecurePOPCSearchNode, VirtualPOPCSearchNode and VirtualPOPCSecurityManager. These modifications will let the future versions of POP-C++ evolve easily one relative to another.

To configuration script will create one of the four follwoing variables : STANDARDSUPPORT, VIRTSUPPORT, SECURESUPPORT, VIRTSECURESUPPORT. The created variable is used in the input makefile (Makefile.am) to define to right compilation process.

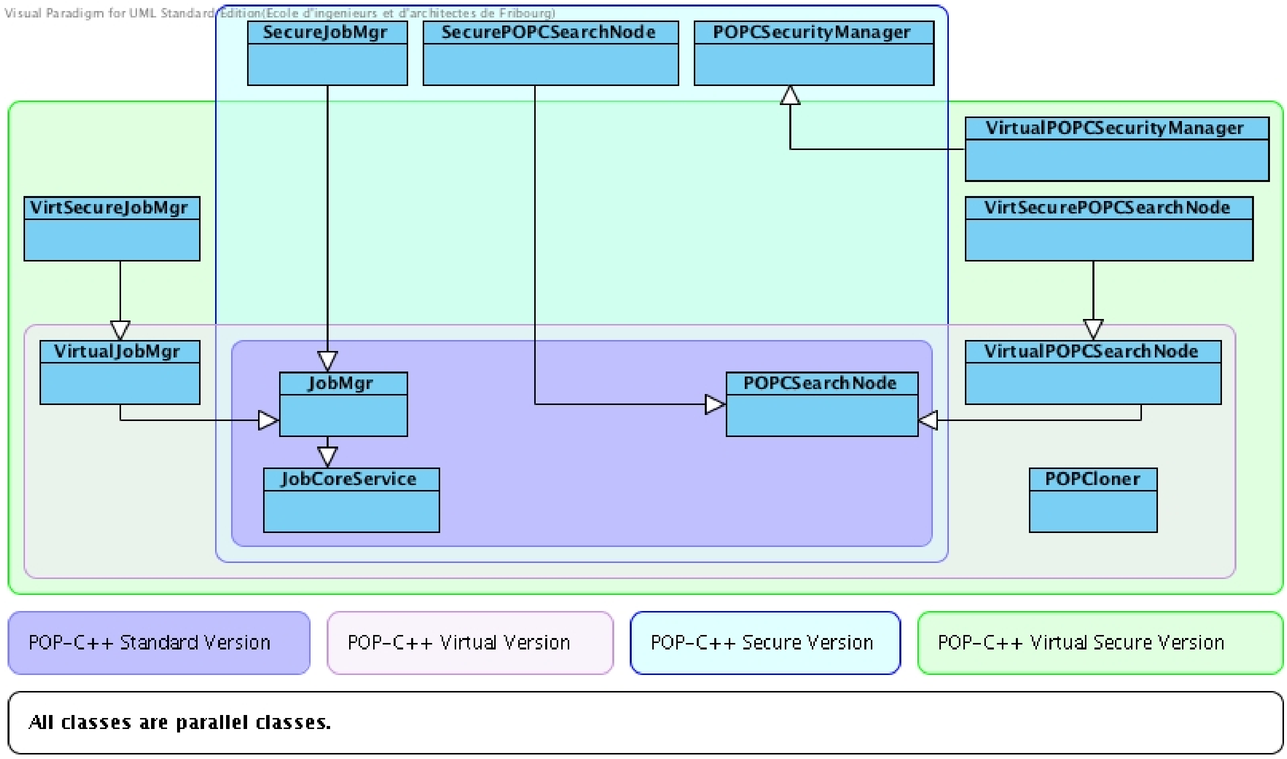
After any modification in the configure.ac file, it is very important to run the following command to generate the new configure script.

```
autoreconf
```

Files created: ./lib/virtual_jobmgr.ph, ./lib/virtual_jobmgr.cc, ./lib/virtual_secure_jobmgr.ph, ./lib/virtual_secure_jobmgr.cc, ./lib/virtual_popc_search_node.ph, ./lib/virtual_popc_search_node.cc, ./lib/virtual_secure_popc_search_node.ph, ./lib/virtual_secure_popc_search_node.cc, ./lib/virtual_popc_security_manager.ph, ./lib/virtual_popc_security_manager.cc, ./services/secure_jobmgr_obj.cc, ./services/virtual_secure_jobmgr_obj.cc, ./services/secure_popc_search_node_obj.cc, ./services/virtual_secure_popc_search_node_obj.cc, ./services/virtual_popc_security_manager_obj.cc

Files modified: ./configure.ac, ./lib/jobmgr.ph, ./lib/jobmgr.cc, ./lib/popc_search_node.ph, ./lib/Makefile.am, ./lib/interface.cc, ./script/popc_setup.in, ./services/Makefile.am

Figure 2: POP-C++ Global Services



5.2 Hypervisor information

Instead of being stored in environment variables, the information relative to the hypervisor connection have been stored in the Virtual JobMgr configuration file. This file is located under *POPC_LOCATION*/etc/virtual.conf. The installation script writes the information directly into this configuration file instead of writing them into a script that initialize the environment variables.

The VirtualJobMgr constructor retrieves all the information stored in the configuration file. These values are stored into attributes and used when needed.

Files modified: ./lib/virtual_jobmgr.cc, ./lib/virtual_jobmgr.ph, ./script/popc_setup.in

5.2.1 Hiding password in POP-C++ Virtual Configuration

The configuration file used in the virtual version of POP-C++ contains some sensible data. These data must be encrypted to avoid unwanted usage of it. To do this task, we choose to encrypt the whole configuration file. This section explains the different algorithm available to this this job and the one we choose in POP-C++ VS.

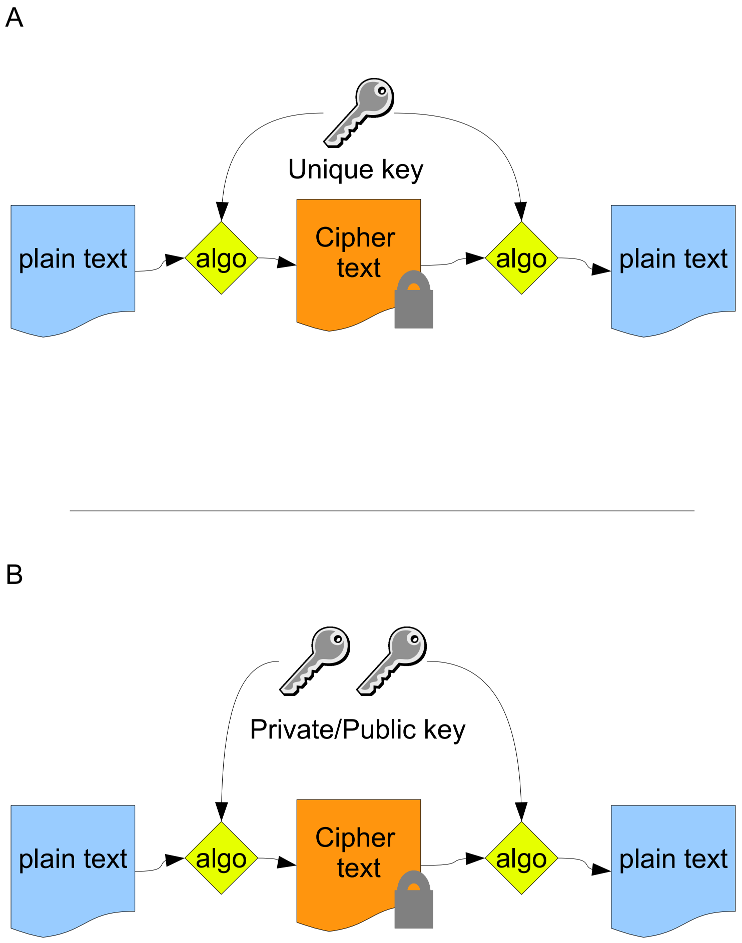
Encryption/Decryption algorithm

There are two main category of encryption/decryption algorithm. These two categories are explained below:

- **Symmetric encryption**: using one key two encrypt and decrypt. Figure **??**A shows the encryption/decryption process of a symmetric algorithm.

- **Asymmetric encryption**: using a pair of private/public keys. One key is used to encrypt the message and the second to decrypt it. Figure 3B shows the encryption/decryption process of an asymmetric algorithm. In this kind of encryption, we could even use an external dongle that manages the encryption and the decryption of the file.

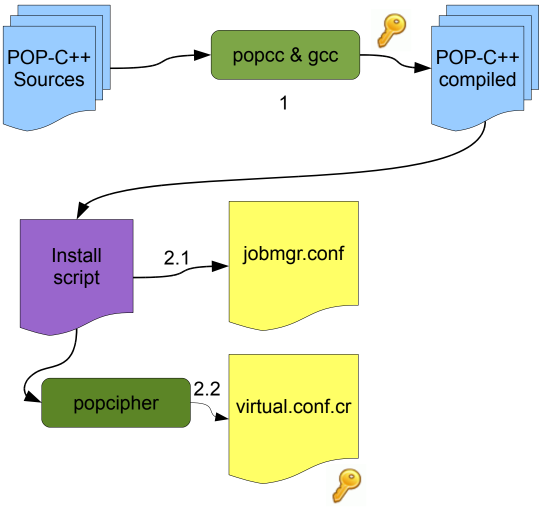
Figure 3: Symmetric and Asymmetric encryption algorithm



Algorithm used in POP-C++ VS

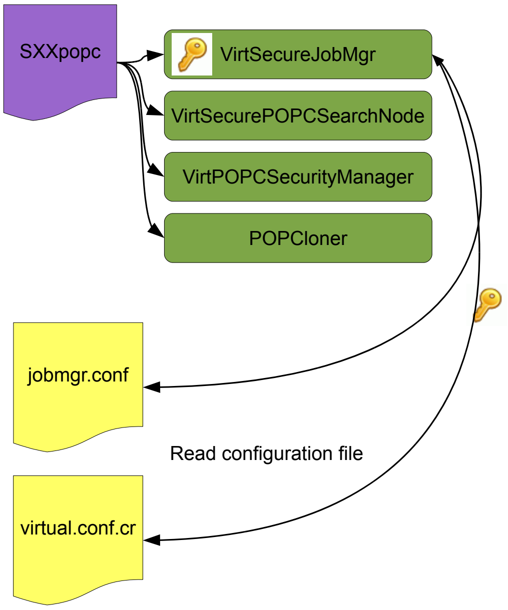
To encrypt the configuration file used in the POP-C++ VS, we choose to use the AES (rijndael) algorithm. The first step is to generate a key and cipher the information about the virtual configuration. Figure 4 shows this first step. During the compilation of POP-C++, a key is generated and integrated to the executable of POP-C++. A small program is compiled to be able to cipher the information about the virtual configuration during the installation of POP-C++. This program is called **popcipher**. As shown in Figure 4, the JobMgr configuration file is not encrypted at all. Only the virtual configuration file is processed by **popcipher** as this file includes some important passwords.

Figure 4: POP-C++ Virtual Configuration - Cipher - Step 1



Once the virtual configuration file is encrypted, POP-C++ need a way to read these information. Figure 5 shows the starts of the POP-C++ Global Services. Once the Virtual Secure JobMgr (VirtSecureJobMgr) is stared by the "SXXpopc" script, this object will read the virtual.conf.ci file. This object has the ability to decipher this file as during the compilation of POP-C+, the generated key has been included in its executable. The VirtSecureJobMgr deciphers the virtual.conf.ci file and uses its information directly in memory. All the information located in the jobmgr.conf file are read as usual as they are all in clear.

Figure 5: POP-C++ Virtual Configuration - Cipher - Step 2



5.3 POP Application Identifier

The AppCoreService has been modified to generate the POP Application ID. This ID is generated in the constructor of this object. The "Request" object has also been modified to include this ID. Just before launching the resource discovery, the JobMgr will contact the AppCoreService to know the POPAppID and set it in the request. The other JobMgrs contacted will know for which application a request has been launched.

For more information about the POP Application Identifier, please refer to the document "POP-C++ over SSH Tunnel"[2] in section 7.1.

In addition to these modifications, the reservation process must also include the POPAppID and the RequestID. The "struct Resources" defined in **./lib/jobmgr.ph** have been modified to include the POPAppID and the RequestID. The "Reserve" method must now accept the POPAppId and the RequestID as parameters. At this point, the JobMgr has all the information to manage the different Worker VM.

Files modified : ./include/request.h, ./include/Makefile.am, ./lib/request.cc, ./lib/appservice.ph, ./lib/appservice.cc, ./lib/jobmgr.cc, ./lib/Makefile.am

Files added : ./include/popwayback.h, ./lib/popwayback.cc

5.4 Modification of the VPopCWrapper

The VPopCWrapper was designed to use only one worker VM at the same time. To be able to manage more than one worker, the VPopCWrapper has been changed. A POPvm object has been created to encapsulate all the information relative to a Worker VM. This object is then passed to some of the wrapper method. Each POPvm object has a pointer to its own domain (pointer of domain is a notion introduced by libvirt). This pointer is saved and then any operation of the wrapper can be executed on this VM.

The POPvm object encapsulates the following information at the moment :

1. The VM name (displayName)
2. The VM configuration file (VMX file on ESXi)
3. The VM volume (datastore on ESXi)
4. The VM OS
5. The VM clean snapshot
6. The VM IP address
7. The POPAppID (if the VM is reserved)
8. The VM PKI
9. The VM state (free, reserved, busy)

The VPopCWrapper interface has been changed to use this new object and be more flexible. The new interface looks as follows :

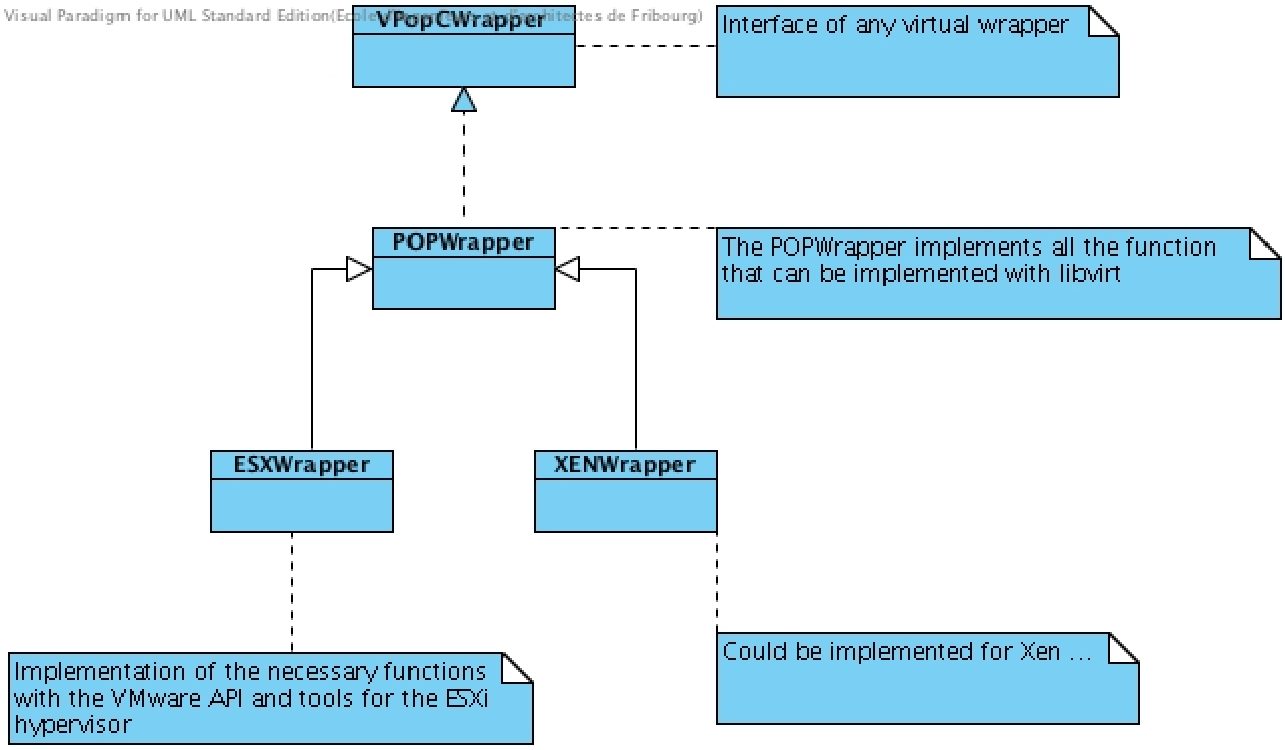
- int _popc_connectOpenAuth(string uri, string user, string password);
- int _popc_connectClose();
- virDomainPtr _popc_domainLookupByName(string domName);
- int _popc_domainState(POPvm &vm, string* state);
- long long _popc_nodeGetFreeMemory();
- int _popc_nodeGetinfo(nodeInfo* info);
- int _popc_domainSnapshotNum(POPvm &vm);
- int _popc_domainSnapshotListNames(POPvm &vm, string names[], unsigned short len);
- int _popc_domainRevertToSnapshot(POPvm &vm, string snapshotName);
- int _popc_domainRevertToOldestSnapshot(POPvm &vm);
- int _popc_domainSnapshotCreate(POPvm &vm, string name, string description);
- int _popc_domainSnapshotDelete(POPvm &vm, string snapshotName, bool removeChildren = false);
- int _popc_domainSetMemory(POPvm &vm, unsigned long memoryInKb);
- int _popc_domainSetMaxMemory(POPvm &vm, unsigned long memoryInKb);

- long _popc_domainGetMaxMemory(POPvm &vm);
- int _popc_domainShutdown(POPvm &vm);
- int _popc_domainSuspend(POPvm &vm);
- int _popc_domainResume(POPvm &vm);
- const string _popc_domainGetMac(POPvm &vm);
- int _popc_domainGetIpAddress(POPvm &vm);
- int _popc_domainIsConnected(string domName="");
- int _popc_sendLocalPublicKey(POPvm &vm);
- int _popc_sendPublicKey(POPvm &vm, POPString pki);
- int _popc_getLocalPublicKey(POPvm &vm);
- int _popc_domainClone(POPvm &original, POPString baseName, int suffix, paroc_accesspoint clonerAP);
- int _popc_domainWaitingForTools(POPvm &vm);
- const string _popc_getError();

In addition to the modifications above, some of the methods in the ESX implementation of the wrapper have been modified to use the VIX API instead of launching shell command. These changes are explained in Section 5.5.

A basic implementation of the VPopCWrapper is done in with the object POPWrapper. This object implements all the methods that can be implemented with libvirt. The ESXWrapper extends the POPWrapper and re-implements only the needed methods. Figure 6 shows the class diagram of wrapper objects.

Figure 6: Virtual Wrapper - Class Diagram


VPopCWrapper
Interface of any virtual wrapper
POPWrapper
The POPWrapper implements all the function that can be implemented with libvirt
ESXWrapper
XENWrapper
Implementation of the necessary functions with the VMware API and tools for the ESXi hypervisor
Could be implemented for Xen ...

5.5 POP-C++ and VIX

As said in Section 5.4, the VIX API is used for some methods in the ESX implementation of the wrapper. These methods were using shell command to do their tasks. It's way better to use API call to perform those actions. Libvirt could one day implement the needed functions available in VIX. If so, the ESX implementation should be modified.

5.5.1 Include VIX in the compilation process

All the necessary modifications done in the configuration script and makefile to include VIX in the compilation process are listed below.

File: configure.ac:
As the VIX API does not install a ".pc" file for the pkg_config function. We have to declare the _CFLAGS and the _LIBS marcos for VIX. The lines below have been added to the "configure.ac" file.

```
VIX_CFLAGS="-lvixAllProducts -ldl"
VIX_LIBS="-I/usr/include/vmware-vix"
AM_SUBST([VIX_CFLAGS])
AM_SUBST([VIX_LIBS])
```

File: ./lib/Makefile.am:
The input Makefile located in the "lib" directory must also be modified. The VIX macros must be added to the macro "AM_CXXFLAGS".

```
AM_CXXFLAGS= OTHER_FLAGS $(VIX_CFLAGS) $(VIX_LIBS)
```

With those modifications, POP-C++ can now be compiled with the VMware VIX API.

NOTE: The VIX API must be installed on the machine which compile POP-C++ in its Virtual or Virtual-Secure version.

5.5.2 Using VIX to get the worker VM IP address

The VIX API allow us to read some properties on the guest VM. This ability is used to read the IP address of the guest VM. The process is going as follows:

1. Connect to the hypervisor (VixHost_Connect())
2. Get a pointer to the VM (VixHost_OpenVM())
3. Power on the VM if not running (VixVM_PowerOn())
4. Waiting for the VMware tools to be ready (VixVM_WaitForToolsInGuest())
5. Read the variable "ip" (VixVM_ReadVariable())

NOTE: For the full code, please see the file ./lib/ESXWrapper.cc method "_popc_domainGetIpAddress(POPvm &vm);"

5.5.3 Using VIX to send and write the admin VM public key on worker VM

The VIX API has two very interesting functions that allow us to send a file on the worker VM and to execute a command on the worker VM. To transfer the Admin VM PKI to the Worker VM, we can send the public key file and write it in the authorized_keys file on the Worker VM by executing a shell command. The process to do this using VIX is going as follows:

1. Connect to the hypervisor (VixHost_Connect())
2. Get a pointer to the VM (VixHost_OpenVM())
3. Power on the VM if not running (VixVM_PowerOn())
4. Waiting for the VMware tools to be ready (VixVM_WaitForToolsInGuest())
5. Login in the guest VM (VixVM_LoginInGuest())
6. Copy the Admin VM public key file to the guest VM (VixVM_CopyFileFromHostToGuest())
7. Execute a command on the guest to write the key (VixVM_RunProgramInGuest())

NOTE: For the full code, please see the file ./lib/ESXWrapper.cc method "_popc_sendLocalPublicKey(POPvm &vm);"

5.5.4 Using VIX to execute a ping on the Worker VM

As the Worker VM is reverted from a snapshot, its saved IP address could be used by another computer when reverted. To be sure to have a valid IP address before executing jobs on the Worker VM, the guest will ping the Admin VM. This procedure will change the guest IP address if this one is invalid on the network. To execute this action, the wrapper uses the VIX API as well. The process is going as follows:

1. Connect to the hypervisor (VixHost_Connect())
2. Get a pointer to the VM (VixHost_OpenVM())
3. Power on the VM if not running (VixVM_PowerOn())
4. Waiting for the VMware tools to be ready (VixVM_WaitForToolsInGuest())
5. Login in the guest VM (VixVM_LoginInGuest())
6. Execute a ping on the guest to contact the Admin VM (VixVM_RunProgramInGuest())

NOTE: For the full code, please see the file ./lib/ESXWrapper.cc method "_popc_reversePing(POPvm &vm, POPString ipAdress);"

5.6 Cloning function implementation

As no built-in cloning function is currently implemented either in the VIX API or in libvirt, this function has been implemented from scratch. Cloning a VM takes quite some time. Due to that, a parallel object will be in charge of the whole process and let the others Global Services do their respective tasks. The POPCloner object is launched at the Global Services start up in the Virtual and Virtual-Secure version. This object is initialized with the ESXi hypervisor information.

When the Admin VM needs to launch a cloning process, a free Worker VM will be reserved as the original VM. This original VM can be used to clone more than one VM. The Admin VM will do a call to the POPCloner and gives the information relative to the orginal VM. The original VM will be busy during the whole process. Once the cloning process is done, the new VM will be added to the list of managed VM and the original VM will be freed. Unfortunately, the cloning process needs several proprietary tools from VMware. Here is the list of those tools :

- VMware VIX API 1.10.2 or later
- VMware CLI 4.1 or later
- VMware tools 8.3.2 or later

The cloning process is detailed below and each action associated with the needed tool/command.

1. Create a new POPvm object with the information of the new VM.
2. Check if the original is suspended or running and shut it down if it's the case. The original VM **must** be shutdown to perform the cloning actions (VIX).
3. Retrieve the configuration file (.vmx) of the original VM (vmware-cmd).
4. Create the new folder for the cloned VM (vifs).
5. List all the files in the original folder and filter them (vifs).
6. Copying files from the original to the new created folder (vmkfstools and vifs).

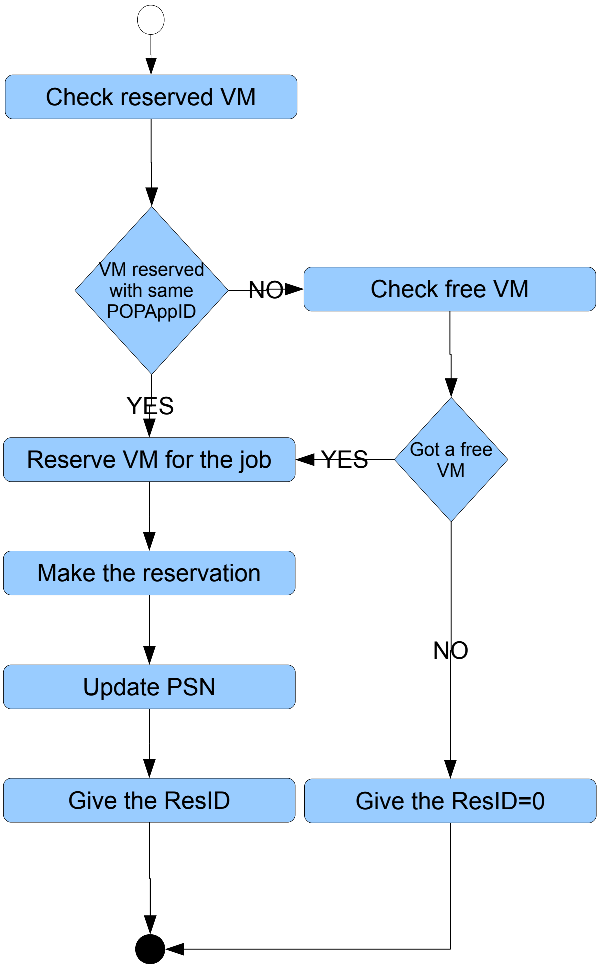
   The .vmx file is read and a new one is created for the new VM.
7. Register the cloned VM (vmware-cmd).
8. Remove useless temporary files
9. Write the VM information in the Virtual JobMgr configuration file.
10. Start and stop the cloned VM (VIX). This step is fundamental to be able to retrieve the VM with libvirt.
11. Add the new VM in the list of usable VM (POP-C++ call to the JobMgr).

NOTE: For the full code, please see the file ./lib/popcloner.cc.
On going development
Even this function is fully functional, we are currently trying to implement this process directly into the libvirt library. As it is still under development, no information can be given now.

5.7 Reservation process

The original reservation process of POP-C++ does not take into consideration the VM. To be able to reserve a VM for a specific POP-C++ Application (POPAppID), the "Reserve()" method has been reimplemented for the Virtual and Virtual-Secure JobMgr. Figure 7 shows a flowchart of the reservation process for both Virtual and Virtual-Secure version.

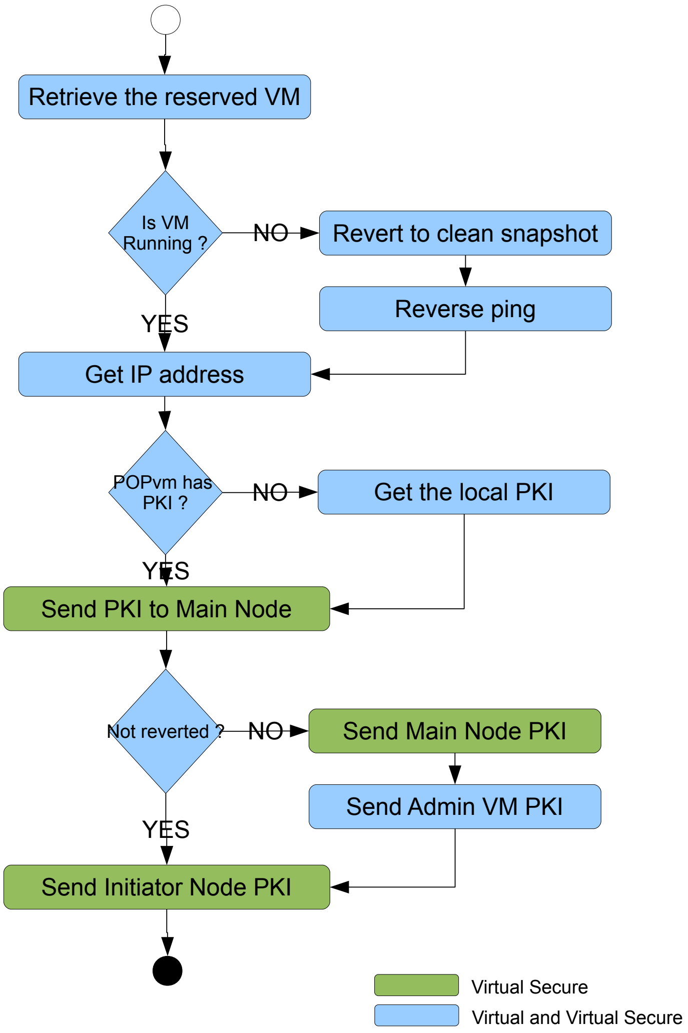
Figure 7: Reservation process



5.8 Preparing a VM to execute a Job

As POP-C++ is now Virtual and Secure, the preparation of the VM to execute a job is a little bit different. Figure 8 shows the flowchart of the VM preparation before the job execution.

Figure 8: VM Preparation



5.9 How to know the Global Services that start an object

A Worker VM do not have Global Services running on it. When an object is created from another object located on the Worker VM, the interface of this worker needs to know which Global Services has created its broker. This information is fundamental to use the key routing process implemented in the POP-C++ Security Manager as the creator node is the node that will write and send the PKIs. This information is known by the JobMgr just after the resource discovery. To be able to give this information back to the Interface, the method "CreateObject()" of the JobCoreService has been changed a little bit. Here is the new signature of this method.

```
sync conc virtual int CreateObject(paroc_accesspoint &localservice, const
paroc_string &objname, const paroc_od &od, int howmany, [in, out,
size=howmany] paroc_accesspoint* jobcontacts, int howmany2, [in, out,
size=howmany2] paroc_accesspoint* remotejobcontacts);
```

The array "remotejobcontacts" will hold the JobMgr access point that created the object as the array "jobcontacts" hold the access point to the object itself.

NOTE: The parameter "howmany2" is not mandatory in pure POP-C++ as "howmany" could be used for the second array size. This parameter has been added to keep the compatibility with POP-Java.

5.10 Virtual POP-C++ Security Manager

The POP-C++ Security Manager (PSM) developed in "POP-C++ over SSH Tunnel" is not able to manage a "virtual node" with an Admin VM and some Worker VM. To manage the security of the virtual node, the Virtual POP-C++ Security Manager has been developed. This parallel object inherits from the orginal PSM.

In POP-C++ VS, the VirtualPOPCSecurityManager (VPSM) works as a router for the PKI. This parallel object will receive request from others nodes in the GRID but also from the Worker VM associated on its own node. With the POPAppID, the VPSM is able to operate a kind of key routing. This chapter explains the modifications done for this routing process.

5.10.1 Add new mapping for Request PKI

When the Admin VM sends a resource discovery request, all the possible resources will answer with their respective PKI. Unfortunately, the VM that will execute this job is maybe not defined yet. To be able to send the PKI of the chosen resource the the VM, a new mapping has been added in the Virtual PSM. This mapping maps a combination of the POPAppID and the ReqID with the PKI.

When a node receives a resource discovery request, it saves the PKI in the mapping. If this node is asked to execute the job, the PKI will be retrieved and sent to the VM.

Files modified: ./lib/virtual_popc_security_manager.ph, ./lib/virtual_popc_security_manager.cc, ./lib/virtual_secure_popc_search_node.cc

5.10.2 Write the key on a VM

When a key reaches the PSM, this key must be written on the right VM. As each running Worker VM is attached with a POPAppID, it's easy to find the right VM to write a key if we know the POPAppID associated with this key.

Once the VM is retrieved, the VPopCWrapper method "_popc_sendPublicKey(POPvm &vm, POPString pki);" is used to write the key on the Worker VM.

Files modified: ./include/VPopCWrapper.h, ./include/ESXWrapper.h, ./lib/ESXWrapper.cc, ./lib/virtual_popc_security_manager.ph, ./lib/virtual_popc_security_manager.cc

5.11 Application ending

The POP-C++ runtime must handle the end of a POP-C++ application. The application services destruction can inform the of the end of a POP-C++ application. Once the destructor of the AppCoreService will be called, a call will be propagated to the JobMgr's method "ApplicationEnd(POPString popAppId, bool initiator)". This method will just remove the associated reservation in the standard version. In the Virtual and Virtual-Secure version, this method will suspend any VM reserved for this POPAppID.

Files modified: ./lib/appservice.cc, ./lib/jobmgr.ph, ./lib/jobmgr.cc, ./lib/virtual_jobmgr.ph, ./lib/virtual_jobmgr.cc

5.12 Unique starting Worker VM

As we are able to clone VM, the assumption of a unique worker VM at first start-up is made. On the first start-up, depending on the configuration entered by the user during the installation, some VM can be cloned. This new VM will not appear in the virtual configuration file as we can find them at global services start-up.

To be able to do this task, the first VM must respect an important rule. The name of the first worker VM must finish with **_worker1**.

5.13 Using valgrind to detect memory leaks

Valgrind is a set of tools to detect incorrect memory management in a program. We try to run the POP-C++ Global Services through valgrind to remove any possible leaks or incorrect memory management.

Running the Interface-side with valgrind
As POP-C++ program are divided into several parts, each part must be run with valgrind. For the Interface-side, the "popcrun" script must be modified to included the valgrind command and options. Here are the modifications made to run the Interface-side with valgrind :

```
cmd="valgrind --log-file=/tmp/$PROG$$ $PROG $args -initparoc
-appservicecode=${MYAPPSERVICE} ${MY_PROXY} -codeconf=${PATHCONF}
/${FILECONF} -jobservice=${MYJOBSERVICE} ${localFlag}"
```

Some leaks have been detected by using valgrind but not all of them are currently corrected. The results below is the last error to be corrected.

```
==19039== Syscall param write(buf) points to uninitialised byte(s)
==19039==    at 0x4062523: __write_nocancel (syscall-template.S:82)
==19039==    by 0x80ACE12: paroc_buffer_raw::Send(paroc_combox&,
   paroc_connection*) (buffer_raw.cc:412)
==19039==    by 0x809F325: paroc_interface::paroc_Dispatch(paroc_buffer*)
   (interface.cc:1167)
==19039==    by 0x808C0F3: AppCoreService::_paroc_Construct(paroc_string
   const&, bool, paroc_string const&) (_paroc3_appservice.ph.cc:28309)
==19039==    by 0x808C5C4: AppCoreService::AppCoreService(paroc_string
   const&, bool, paroc_string const&) (_paroc3_appservice.ph.cc:28291)
==19039==    by 0x80923C7: paroc_system::CreateAppCoreService(char*)
   (system.cc:477)
==19039==    by 0x80931B3: paroc_system::Initialize(int*, char***)
   (system.cc:374)
==19039==    by 0x8091F0A: main (paroc_infmain.std.cc:53)
==19039==  Address 0x430f971 is 281 bytes inside a block of size
   1,044 alloc'd
==19039==    at 0x4025BD3: malloc (vg_replace_malloc.c:236)
==19039==    by 0x4025C5D: realloc (vg_replace_malloc.c:525)
==19039==    by 0x809881F: paroc_array<char>::SetSize(int)
   (paroc_array.h:156)
==19039==    by 0x80ACCB2: paroc_buffer_raw::Reset() (buffer_raw.cc:30)
==19039==    by 0x80ACFEA: paroc_buffer_raw::paroc_buffer_raw()
   (buffer_raw.cc:19)
==19039==    by 0x80AD11C: paroc_buffer_raw_factory::CreateBuffer()
   (buffer_raw_factory.cc:23)
==19039==    by 0x809FCAF: paroc_interface::Encoding(paroc_string)
   (interface.cc:800)
==19039==    by 0x80A0089: paroc_interface::NegotiateEncoding(
   paroc_string&, paroc_string&) (interface.cc:886)
==19039==    by 0x80A223B: paroc_interface::Bind(char const*)
   (interface.cc:606)
==19039==    by 0x80A294F: paroc_interface::Bind(paroc_accesspoint const&)
   (interface.cc:480)
==19039==    by 0x80A4ACF: paroc_interface::Allocate() (interface.cc:444)
==19039==    by 0x808C5A6: AppCoreService::AppCoreService(paroc_string
   const&, bool, paroc_string const&) (_paroc3_appservice.ph.cc:28290)
```

Running the Broker-side with valgrind

To run the Broker side with valgrind, we need to change the To run the other side of a parallel object using valgrind, the execution of a POP-C++ application was set to manual. In this state, the POP-C++ runtime just print the command used to run the parallel object on the other node. By modifying this command, we are able to run the parallel object broker-side with valgrind.

The result of this execution presents no leaks in memory.

6 General modifications

This chapter regroups all the modifications made on POP-C++ in general. Some of these modifications have been made to fix bugs and some to enhance to possibilities of POP-C++ in all versions.

6.1 Update the POP-C++ Search Node resource information

Currently, the POP-C++ Search Node (PSN) is not updated when a reservation is made on the node. To fix this bug, the PSN has now two additional methods to add a job and to remove a job. With this update, the PSN can directly refuse a resource discovery and not send a response and refuse the reservation later.

These methods increase or decrease the resources of the PSN and the number of job currently executed on the node. Here are the signature of these two methods :

```
sync seq void addJob(float power, float memorySize, float bandwidth);
sync seq void removeJob(float power, float memorySize, float bandwidth,
     int nbJob);
```

To reduce the number of call to the PSN, the removing method is called once for all the application.

7 Security assumptions

This chapter aims to explain what we think are strengths and weaknesses on a security point of view in this project. The first section will focus on the strength as the second one will focus on the weaknesses.

7.1 Security strengths

7.1.1 Application isolation

An application is isolated from other application as a Worker Vm is started for each different application on a node. This means that an application can only disturb itself but not other running application.

7.1.2 Admin VM isolation

The Admin VM is totally isolated from the execution of the POP-C++ application. This VM do not accept other public key than the ones exchanged for the confidence link. An application should never be launched from an Admin VM.

7.1.3 Worker VM clean state

A Worker VM is always reverted to its clean state before executing an application. Of course, this clean state depends on the person who created it during the installation of the node.

7.2 Security weaknesses

7.2.1 Node running the main

Every POP-C++ application has a start point which is the main. The main must run on a node we will called the Main Node. The Main Node could be vulnerable has it will accept several public key from the worker used for this application. For the moment, we did not develop a perfect solution for this problem but some ideas are drawn in Section 9.

7.3 Possible security attacks

This section aims to expose some of the possible attacks we found possible to happen on our middle-ware and the way to avoid them.

7.3.1 Sniffing the network

By sniffing the network, the possible hacker could not do anything to our confidence network. Every connections are encrypted into a SSH Tunnel.

7.3.2 Hacking the Admin - VM

An external person could hack into the Admin VM. As the Admin VM just acts as a routing point for the key exchnage,

7.3.3 Enter the confidence network with a fake POP-C++

A malicious developer could modify POP-C++ before installing it. This means that a different version of POP-C++ will enter a confidence network with standard POP-C++ VS. This kind of hack could disturb the good working of an execution. As the key transferred between the node are the public key, they do not pose a security issues if they are intercepted.

7.3.4 Make a POP Application with back doors

A developer could make a POP Application that keep some process alive after its execution. As only one application is running in a Worker VM and these Worker VM are reverted at end, this kind of hack pose no problem to our middle-ware.

8 Test

This chapter explains all the tests done on the new version of POP-C++ to be sure that this version is functional. The second part of this chapter is a report on the performance of the new version.

8.1 Functional tests

With the key exchange process, some functional tests have been made to verify the correct behaviour of this key exchange. These tests are the same as the ones explained in the "User Manual" add-on.

8.1.1 Real situation test

To prove our theory, a real test situation has been put in place at EIA-FR. This section will explain the infrastructure and the constraints of this situation.

For the moment, this test couldn't have been done as Geneva did not set up a correct POP-C++ VS Node for the moment.

EIA-FR test infrastructure
The GRID & Cloud Computing Group has got a DMZ that can be managed without the intervention of the IT Head Office of the school. This DMZ is delimited from the Internet by a firewall (FW). This FW is able to manage different areas. These areas are working as different networks. Nowadays, four different areas are in place.

- "Grille Collaborateurs" - Collaborators GRID
- "Grille étudiants" - Students GRID
- "Servers"
- "Grille Test" - Test GRID

Figure 9 shows the current configuration of the EIA-FR DMZ
The subnet 160.98.22.0/23 is available for the computers located in the DMZ.

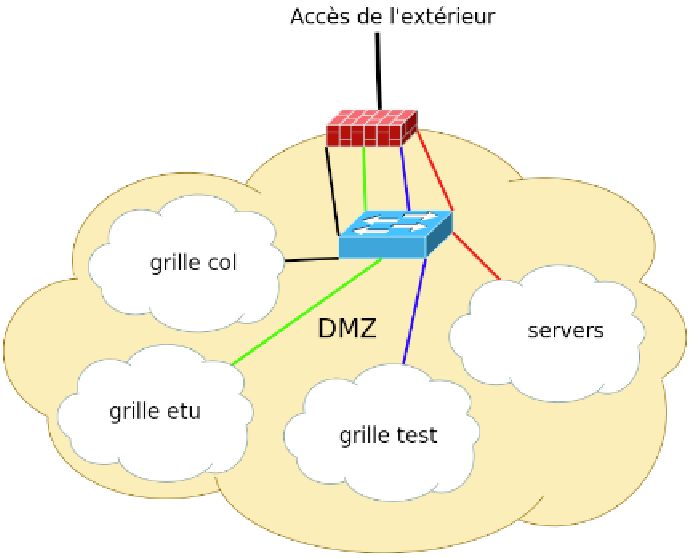
Restriction
The only restriction imposed by the IT Head Office is to block the SMPT port (25) and the SMPT-S port (587). This restriction avoids the unwanted use of a mail server installed in the DMZ as a routing server for SPAM. This problem could put the all school network (160.98.0.0) to be on a blacklist.

Area
The IP address range 160.98.22.1 to 160.98.22.254 is subdivided as follows:

Area	**Subnet**	**Gateway**	**Mask**
Grille étudiants	160.98.22.0	160.98.22.1	255.55.255.192
Grille collaborteurs	160.98.22.64	160.98.22.65	255.55.255.192
Grille test	160.98.22.128	160.98.22.129	255.55.255.192
Servers	160.98.22.192	160.98.22.193	255.55.255.192

Rules have been globally defined for each area. Some specific rules are defined for well defined computers.

Figure 9: EIA-FR DMZ



Proposition for the ViSaG project
To avoid external access to the internal school network, the school network is simulated by an area in the DMZ.

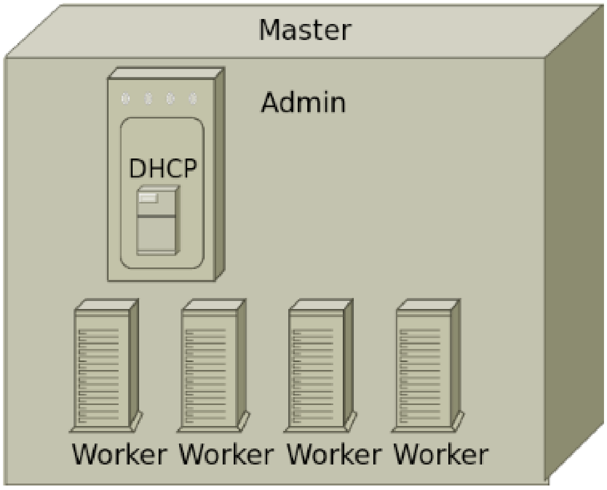
A computer named "Master1", in the rest of this section, will be installed in the area "Grille collaborateur". In the ViSaG project, this computer will be the only access point for HEPIA and HE-Arc.

A second computer named "Master2" will be installed in the area "Grille Test". This computer will be accessible by Master1. Master1 and Master2 are physical computer running VMware ESXi.

On these computers, a virtual machine (VM) named "Admin" will run permanently. The "Admin" VM on Master1 and the one on Master2 will be able to communicate together because they made the confidence link. When a request is coming on one "Admin" VM, a "Worker" VM is launched. The computation work will be done in these "Worker" VMs.

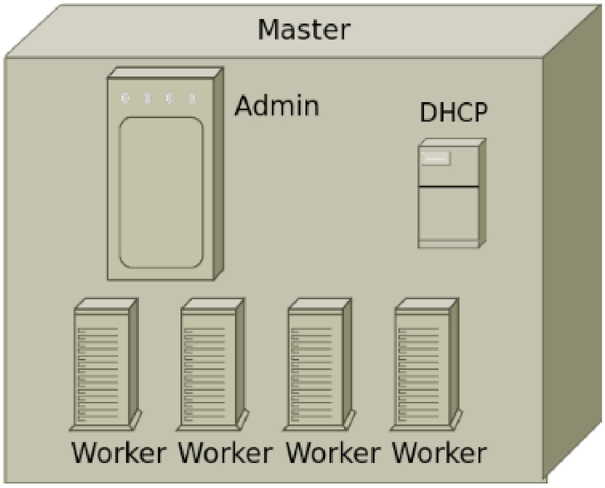
A DHCP server needs to be set to give IP Address to the "Worker" VM. As shown on Figure 10, the DHCP server will be installed on an "Admin" VM. The IP address distributed by the DHCP server must be public address.

Figure 10: DHCP server on "Admin"



The DHCP server could also be set on an external VM as shown on Figure 11.

Figure 11: DHCP server on an external VM



Important point

It is important than only one DHCP server is accessible in an area. The FW will be configured to communicate with a specific computer with a specific IP address. If the "Worker" get his IP address from another DHCP server, this address will be blocked by the FW.

In order that the "Worker" managed by the "Admin" on Master1 can communicate with the "Worker" managed by the "Admin" on Master2, the port 22 (SSH) must be open for the IP address attibuted to the "Worker". A number of rules egal to the number of "Worker" + 2 ("Master" + "Admin") must be defined on the FW. To avoid to many rules on the FW, the DHCP server will be configured to attribute only the good number of IP address for the "Worker" VM.

Figure 12: Logical View of the infrastructure

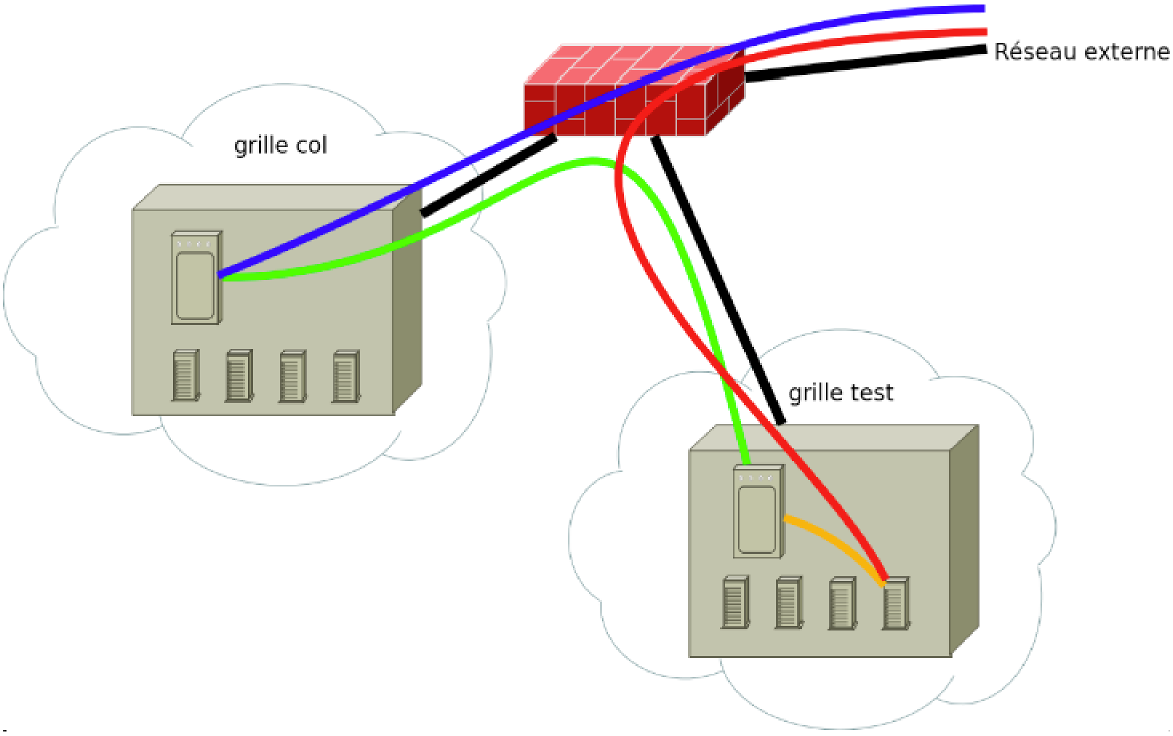


Figure 12 shows a logical view of the DMZ with our two different areas working as two different networks separated by a FW. The different link colors represent:

- Black: physical connection
- Blue: connection on the "Admin" VM
- Green: Communication between JobMgr
- Orange: "Worker" creation
- Red: Communication between "Worker" and the POP-C++ application

8.2 Performance tests

We were aware that using virtual machine and secure connection will affect the performance of POP-C++. To know which part of the execution is the slowest compared to the standard version, some performance tests have been made. This section explains the different test case and the results.

8.2.1 Matrix Computation with POP-C++ Standard Version

To have a basis, we first made performance test on the same architecture but with the Standard version of POP-C++. This section reports all the measures and the results.

Infrastructure
The infrastructure is composed by three Virtual Machines running on three different ESXi platforms. Each VM have a standard version of POP-C++ installed. The configuration of the global services is made to share the charge between the three nodes.

Measurements

As the infrastructure for the POP-C++ VS is small, the test for the standard basis version must be done on the same small infrastructure of three nodes. The tests have been performed with three different matrix size and for each matrix size, three different numbers of workers.

Matrix Size	Nb Workers	Nb workers/VM
600x600	1	1
600x600	3	1
600x600	6	2
600x600	9	3
900x900	1	1
900x900	3	1
900x900	6	2
900x900	9	3
1200x1200	1	1
1200x1200	3	1
1200x1200	6	2
1200x1200	9	3

Results

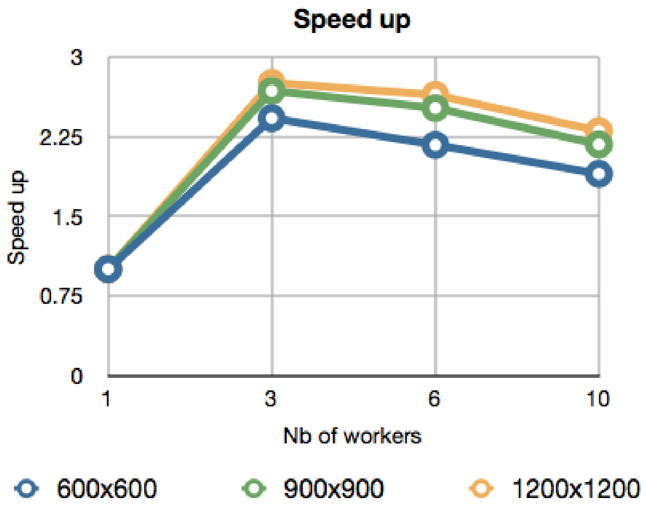
To have a basis for the performance tests on POP-C++ VS, we have run the same tests on the same architecture but with the POP-C++ standard version. This section details the result of these executions.

Speedup

The speedup has been computed with the different results obtained.

Matrix Size/Nb of workers	1	3	6	10
600x600	5.17748	2.1383	2.3859	2.7290
900x900	16.4920	6.1572	6.5519	7.5837
1200x1200	38.5506	13.9942	14.5949	16.7564

Figure 13: Speedup for POP-C++ Standard execution



As we can see on Figure 13, the best speedup is found with 3 workers.

8.2.2 Matrix Computation with POP-C++ Virtual-Secure version

The same tests have been done with the Virtual-Secure version of POP-C++. This section details the result of these execution.

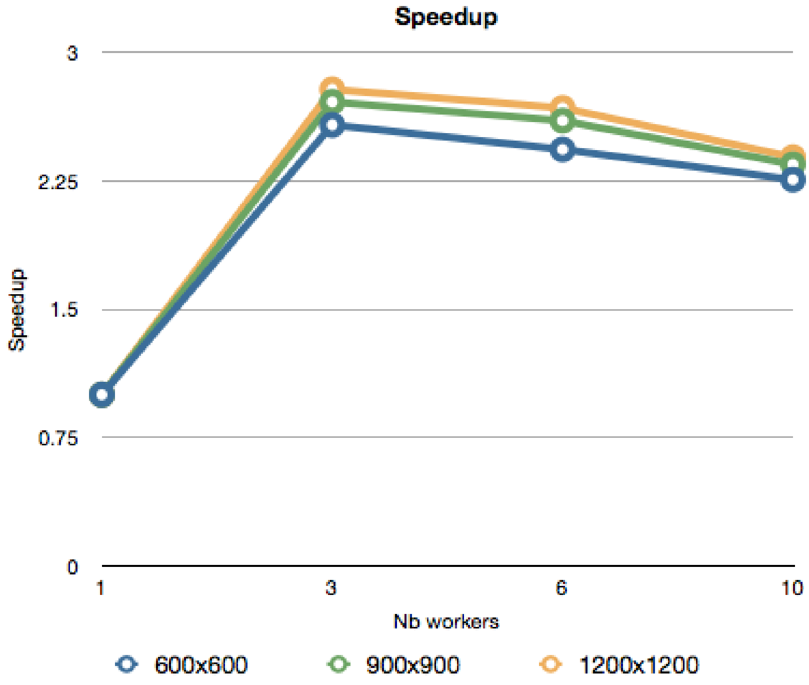
Speedup

The speedup has been computed with the different results obtained.

Matrix Size/Nb of workers	1	3	6	10
600x600	4.838814	1.87972	1.99048	2.145922
900x900	16.22704	5.991882	6.244004	6.923945
1200x1200	38.37998	13.79878	14.35519	16.08427

As we can see on Figure 14, the speedup is quite similar with the standard version because this not take into consideration the initialization time.

Figure 14: Speedup for POP-C++ Virtual Secure execution

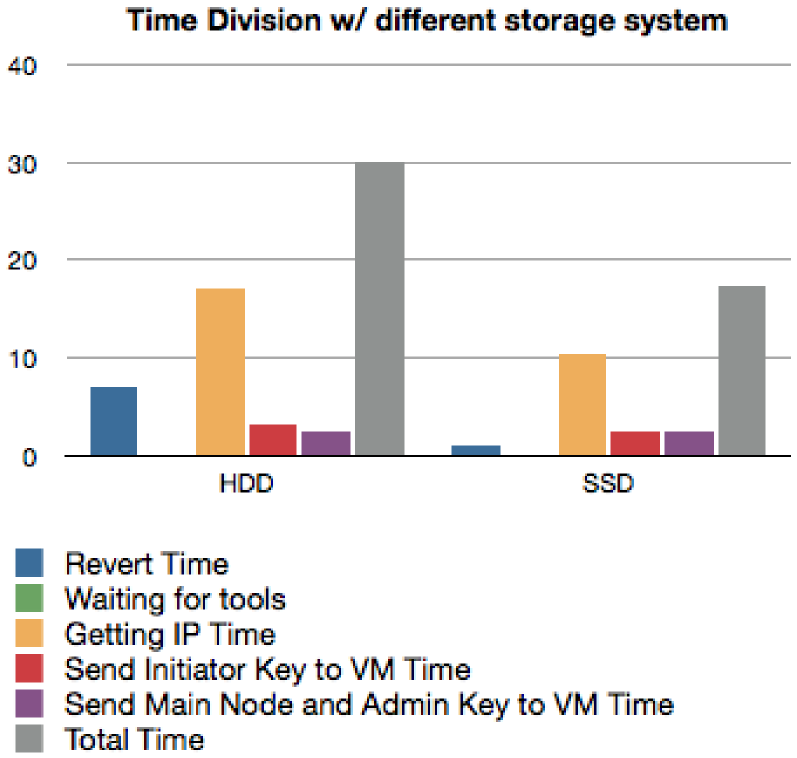


8.2.3 Test with SSD

To try to have better performance, we try to use SSD instead of standard hard drive disk. Here is the result of these tests that have been running on HDD and on SSD.

As shown in Figure 15, using a SSD instead of a HDD, the time to prepare a VM is almost twice smaller. This could really improve the usage of POP-C++ VS as the main time loss is during the VM preparation.

Figure 15: Preparing VM time with HDD and SSD



8.3 Conclusion

As we can see, we do not loose performance during the computation. In fact, all the losses are due to the preparation of the Worker VM. This means that the creation of an object is much slower when a Worker VM has to be prepared. A creation could take about 20 times more if no Worker VM is ready for the object and a whole preparation must be done. This time could be almost reduced by 2 if using a solid storage drive.

9 Future development

This chapter will develop several idea that could be implemented in future development of POP-C++ VS.

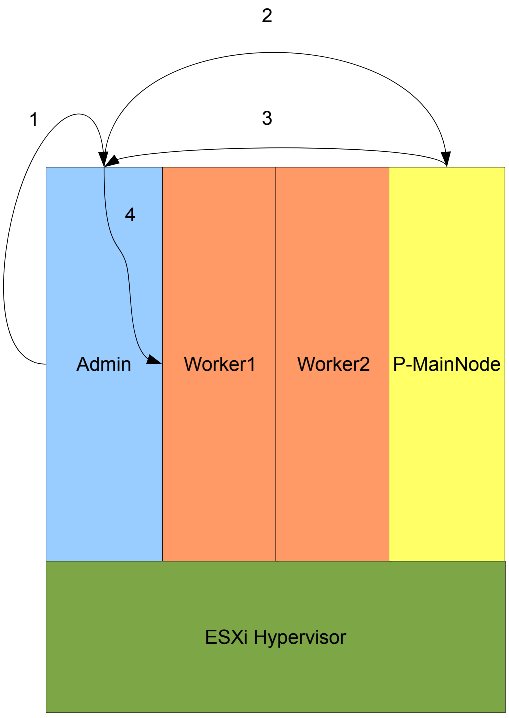
9.1 Pseudo Main Node

As explained in Chapter 7, the node running the main of a POP-C++ application is exposed to some risks. To avoid to use the developer node or an Admin VM as the Main Node, a mechanism of pseudo Main Node could be developed.

The pseudo Main Node should be a VM running POP-C++ Secure and started by the Admin VM. Figure 16 shows a schematic view od a ESXi Node running POP-C++ VS and a VM acting as a pseudo Main Node. This mechanism should work as follows:

1. An application is started on the Admin VM.
2. The Admin VM dected the start of an application, starts the Pseudo Main Node VM and transfer necessary information on this VM. The application is then started from this VM.
3. The Pseudo Main Node asks for resources to the Admin VM.
4. The Admin VM starts a worker to give resources to the application.

Figure 16: Schematic view of the an ESXi Node with "Pseudo Main Node VM"



These steps could be performed by the **popcrun** script or by the paroc_main function.

9.2 Lock Worker VM

As Worker VM will execute unknown executable, it could be a great idea to lock as much as possible the outgoing traffic on this VM. On Ubuntu distribution, a simple command line utility named "ufw" allows to apply rules for outgoing and incoming traffic.

9.3 Keep keys table for each workers

At the moment, each time a key need to be rerouted, it's done. What could be a great enhancement is to keep a table in the JobMgr about each workers and the keys that have been written on this worker. With this kind of table, we will be able to know if a key has to be wrriten on the worker or not.

10 Glossary

- **API** : Application Programming Interface
- **VIX** : API of VMware to interact with the VMware hypervisor (or Workstation ...)
- **EIA-FR** : College of Engineering and Architecture of Fribourg
- **HEPIA** : Haute Ecole du Paysage, d'Ingénierie et d'Architecte de Genève
- **POP-C++** : Parallel Object Programming C++
- **ViSaG** : Virtual Safe GRID
- **VM** : Virtual Machine

11 Table of figures

12 References

[1] Adrian Wyssen, *VirtualPOPC-1 : Project Report*. EIA-FR, Switzerland, June-August 2010.

[2] Valentin Clément, *POP-C++ over SSH Tunnel*. EIA-FR, Fribourg, Switzerland, September-November 2010.

[3] Valentin Clément, *POP-C++ Virtual-Secure : POP-C++ User and Installation manual add-on*. EIA-FR, Fribourg, Switzerland, January 2011.