Horizon 2020
European Union funding
for Research & Innovation

SecureCloud

Joint EU-Brazil Research and Innovation Action
SECURE BIG DATA PROCESSING IN UNTRUSTED CLOUDS

https://www.securecloudproject.eu/

Preliminary implementation of the communication and storage mechanisms
D4.2

Due date: 31 December 2016
Submission date: 24 January 2017

Start date of project: 1 January 2016

Document type: Deliverable
Work package: WP4

Editor: Marcell Feher (CC)

Reviewer: Luigi Romano (SYNC)
Rodrigo Jardim Riella (LACTEC)

Dissemination Level

PU	Public	√
CO	Confidential, only for members of the consortium (including the Commission Services)	
CI	Classified, as referred to in Commission Decision 2001/844/EC	

SecureCloud has received funding from the European Union's Horizon 2020 research and innovation programme and was supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under grant agreement No 690111.

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T4.1	Secure distributed communication mechanisms	UniNE*, IMP, CC
T4.2	Secure distributed data management and storage	CC*, IMP, UniNE, UTFPR

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Contents

1 Introduction

The vision of the SecureCloud project is to enable secure execution of Big Data applications within cloud environments that could be malicious. To achieve this, SecureCloud uses a microservices approach to deliver basic functionalities to larger applications. In particular, SecureCloud's goal is to maintain at a minimum the system components that require secure execution in SGX enclaves, which is also key to executing distributed applications and services that are at the core of Big Data applications. This allows for a leaner design (e.g., less resource use, higher execution speed) that uses secure resources when and where they are critical in the Cloud. This approach is different from other projects, e.g., Microsoft's Haven [2], that seek to execute legacy applications together with a library operating system (OS) inside an enclave, requiring a large trusted computing base as well as a larger overhead to support the library OS, higher penalties to transition to/from the enclave when the OS requires interaction with resources outside the enclave, and the higher enclave memory dedicated to supporting these code base and functionality.

This deliverable describes the basics of two demonstration components built from core microservices developed in WP4, namely, communication and storage. These implementations are developed from the specifications and early implementation versions described in deliverable D4.1. More specifically, this deliverable describes the basics of the demonstrators and the steps necessary to find, compile, build and run such demonstrators.

The demonstrator for secure communication focuses on SCBR (Secure Content-Based Routing). Although, content-based routing (CBR) supports scalable asynchronous communication among large sets of geographically distributed nodes, preserving privacy constitutes one of its main limitations for its widespread use. This is in part due to the fact that a CBR router must see the content of the messages sent by data producers, as well as the filters (or subscriptions) registered by data consumers. SecureCloud follows a different strategy and implemens a CBR engine in a secure enclave provided by Intel's SGX. Thus, the compute-intensive CBR operations can operate on decrypted data shielded by the enclave and leverage efficient matching algorithms.

The demonstrator for distributed storage focuses on an implementation of a secure and distributed key-value store functionality. This is key not only to enable secure processing of Big Data, but to maintain a secure and trusted storage of such data within the cloud architecture. At its core, SecureCloud's design allows the system to process data within secure enclaves and protecting it (e.g., using encrypt, network encode/decode) before storing it within the untrusted storage infrastructure. This is one of the key goals of WP4, and critical to deliver Big Data processing mechanisms, e.g., Map-Reduce, which are meant to operate on massive amounts of data, i.e., which cannot be stored in a single computer and let alone a single SGX enclave. SecureCloud proposes a scalable architecture that separates the Key-Value Store API, the encryption and coding instances (for managing different levels of data encryption and network coding) executed in Intel SGX chipsets, raw data storage management, and the Data Base operations required for accelerating search processes. The implementation is carried out within the OpenStack project and assuming some of its functionalities and components, but the general design is compatible with other Big Data frameworks. This secure implementation of Key-Value store not only provides mechanisms to secure the data, but also to provide a series of storage options that will trade-off storage space (important to reduce system costs), data locality (important for speeding up processing), and data characteristics (hot data versus cold data).

We organize this deliverable as follows. Chapter 2 describes the basics of SCBR implemented by SecureCloud for demonstration and the installation process. Chapter 3 describes the Secure Key-Value Store framework and implementation as well as the process to install and run the demonstrator. Chapter 4 concludes this Deliverable.

2 Secure distributed communications demonstrator

SCBR (Secure Content-Based Routing) forwards messages based on their content. Clients, consumers or subscribers register their interest by providing the conditions a given publication would have to fulfill to be routed to them. SCBR stores those filters inside secure SGX containers. Sources, producers or publishers provide publications that are matched against the stored subscriptions and forwarded to interested clients. Technical details regarding the implementation of the system can be found in papers [1] and [3], and in Deliverable D4.1.

In the demonstrator, we provide the source code for a Consumer, a Source and SCBR implemented according to [3]. The provided software is at an experimental/alpha state, which means that it is still under development. Since it was conceived as proof of concept (PoC), it includes not only functional code but also code to inspect and evaluate its behavior and that of the demonstrator in measurement campaigns. Therefore, in its current form it is not meant to be deployed as production code and it may contain bugs.

The code can be found at `https://gitlab.securecloud.works/consortium/deliverables/tree/master/D4.2_CC/T4.1_Communication_mechanisms_demo`.

2.1 Messages format

Publication and subscription headers are in ASCII format, as follows:

```
0 0 P i symbol 58 i date 1136764800 i high 0 i low 0 i volume 2250

0 8945 S i symbol = 29 i open >= 132 i open <= 427
```

Publications header contain a set of pairs of key and numeric values. The payload of such messages is arbitrary, defined by the application that runs on top of it. Subscription headers, besides key and values, also include a comparison operator (=, >, <, >=, <=). Each field is separated by the letter `i` with spaces before and after.

Datasets for running the applications are included in the folders `scbr/cbr-prefilter/pubs` and `scbr/cbr-prefilter/subs`.

2.2 Compilation

The provided code include Makefiles to build the code. It depends, however, on the compiler `g++-5` and other libraries, as follows:

- ZeroMQ message queue library v. 4.1.4-7 (package `libzmq3-dev`)
- Crypto++ cryptographic library v. 5.6.1-9 (package `libcrypto++-dev`)
- Boost C++ libraries v. 1.58.0 (only `-lboost_system` and `-lboost_chrono`)
- Intel SGX SDK Linux x64 v. 1.7.100.36470

Assuming all dependencies are properly installed, it suffices to type `make` in the `scbr` directory to compile `source` and `consumer` . If no encryption is wanted, one should pass a parameter to the

Makefile, as follows:

```
$ cd scbr
$ make ARGS=plain
```

The SCBR, on the other hand, should be compiled with the provided scripts (`build-encr-sgx.sh`, `build-encr-unshield.sh`, `build-plain-sgx.sh` or `build-plain-unshield.sh`).

```
$ cd scbr/sgx/
$ ./build-encr-sgx.sh
```

It is important to note that if one wants to run on a skylake processor with support to SGX, besides allowing it in the BIOS, it is also needed to set the variable `SGX_MODE` to `HW` inside `scbr/sgx/enclave_CBR_Enclave_Filter/sgx_u.mk` and `scbr/sgx/enclave_CBR_Enclave_Filter/sgx_t.mk`. Otherwise it runs in simulation mode.

2.3 Running

2.3.1 Source / Producer / Publisher

Generated executable: `scbr/bin/source`

`-a`	`--address=host:port`	Where to listen. Default: *:5557
`-i`	`--input=filename`	Publications file
`-k`	`--key=filename`	File containing RSA Private key

For convenience, a pair of keys are packed within the archive.

2.3.2 Consumer / Client / Subscriber

Generated executable: `scbr/bin/consumer`

`-a`	`--address=host:port`	Where to listen. Default: *:5558
`-i`	`--input=filename`	Subscriptions file
`-k`	`--key=filename`	File containing Source (Publisher) RSA Public key

2.3.3 SCBR

Generated executable: `scbr/sgx/enclave_CBR_Enclave_Filter/scbr`

This process can either bind and listen to a port or it can actively connect to listening sources and consumers. Either the option `-a` or a list of options `-c` must be provided, but not both at once.

`-a`	`--address=host:port`	Where to listen. Default: *:5555
`-c`	`--connect=host:port`	Where to connect. Multiple possible.
`-s`	`--silent`	No output
`-v`	`--verbose[=`v`]`	Displays more information on screen. Verbosity increseases if arg is passed

Through the following commands, one should be able to run a test with 10 subscriptions and 1000 publications with the given datasets.

```
$ ./sgx/enclave_CBR_Enclave_Filter/scbr -c localhost:5553 -c
localhost:5551
$ ./bin/source -i cbr-prefilter/pubs/p1k -a *:5553 -k key.prv
$ ./bin/consumer -i cbr-prefilter/subs/s10 -a *:5551 -k key.pub
```

3 Secure distributed data management and storage

As part of the basic infrastructure of the SecureCloud ecosystem, Chocolate Cloud ApS designed and developed the prototype of a general-purpose key-value store solution, which provides confidentiality and reliability guarantees. The former is a result of the process of encryption and/or (network) encoding of the data, which provides a natural and second layer of encryption as described in D4.1, to store sensitive data in different locations/devices within the Cloud securely. The latter comes from the fact that the use of network coding provides not only added confidentiality to the data, but also resilience against the lost of devices and storage locations within the Cloud.

3.1 Deliverable structure

The delivered sources are organized as follows:

- `api-specification`: An interactive REST API specification webapp, that describes in detail all the supported functions and allows the user to try them out
- `cross-component_messages`: Under the hood, the individual services that make up the Secure Key-Value Store communicate with each other using messages serialized using Google's *Protocol Buffers*. These are defined in this folder
- `ec-worker`: Source code of the microservice that performs erasure coding using Chocolate Cloud's technology
- `mock-objectstore`: Source code of a simple objectstore compontent, which is used to persistently store replicated blocks and erasure coded fragments. This little service will be replaced by a real distributed block storage system like *OpenStack Swift* or *CEPH* later
- `rest-frontend`: Source code of a webserver that implements the REST interface and forwards requests to the key-value store underneath
- `secure-key-value-store`: Source code of the main component, the Key-Value Store microservice

Additionally to the source code of the components and inter-component message declarations, we prepared a Virtualbox virtual machine image, where the demo system is fully installed and configured. Information about the demo VM is summarized in Table 3.1.

After launching the virtual machine, a Readme file on the desktop describes the few easy steps to start all system components and the interactive API explorer that allows the user/reviewer to try the system functionality. For convenience, a shell script is added to start the components.

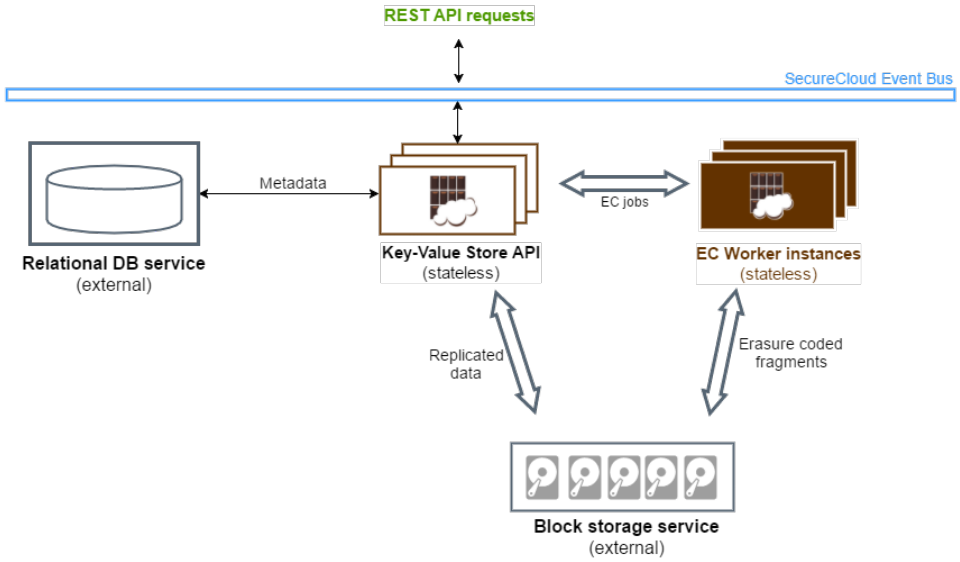
Table 3.1: Key-Value Store demo VM

URL	https://www.securecloudproject.eu/t42-storage-demo/
Total size of the VM	3 093 340 160 bytes (2.9 GB)
VM Platform	VirtualBox
VM Operating System	Ubuntu 16.04 LTS
Password of the VM user	cloud1234

3.2 System Overview

The preliminary implementation is a single-machine demo of the distributed key-value store that we are aiming for in the final SecureCloud framework. The conceptual architecture of the storage service is shown on Figure 3.1.

Figure 3.1: Secure Key-Value Store service conceptual architecture



API requests are processed by the component labelled *Key-Value Store API*. It has an understanding of Objects, Datastores, Storage Policies, and all other entities of the interface, and implements most of the functionality of the system. It uses the other three components when serving incoming requests: a relational database is used to store and retrieve system metadata (e.g., hierarchy of Objects of Datastores), a set of workers are used to perform Erasure Coding, and a block storage service is responsible for storing replicas and erasure coded fragments persistently.

These components are designed to be ran either on shared or separate nodes of the system. The data channels between them are implemented using ZeroMQ, which offers an abstract socket to the programmer, effectively allowing the components to scale seamlessly between running as threads, processes on a single machine, or on different machines with network connection between them. This

ensures that, even though the shipped deliverable is a single-machine demonstration of the storage system, it will be used in a real-life environment with numerous nodes.

4 Summary and Conclusions

In this deliverable, we have presented the basic description and details for accessing and running the demonstrators for secure and distributed communication and storage, which focus on SCBR and Secure Key-Value Store. Inherently, the problem of CBR and Key-Value Store are of great interest and importance to Big Data applications and Cloud computing, but the ability to ensure security (e.g., confidentiality, privacy) of the highly sensitive data depended on trusting Cloud providers, which could be compromised. SecureCloud's focus allows the trust to be shifted to the underlying hardware, while maintaining the key features that make these solutions attractive. These two components rely on an implementation of the proposed architectures and microservices described in D4.1 and constitute a first system implementation of WP4, particularly Tasks 4.1 and 4.2, that can be used by larger Big Data processing applications.

Following SecureCloud's spirit, SCBR and Secure Key-Value Store rely on highly scalable microservices, naturally dividing the process of communication and storage in different components/steps that can be run in parallel and scaled independently, while also identifying where and what part should be run on a secure enclave. The underlying microservices provided are at an experimental/alpha state of development as a result of the work in this first 12 months of work in SecureCloud. Further development of these microservices, the demonstrators, and their integration into larger applications for Big Data processing will be the focus of WP4's members and the SecureCloud consortium.

Bibliography

[1] R. Barazzutti, P. Felber, H. Mercier, E. Onica, and E. Rivière. Thrifty privacy: Efficient support for privacy-preserving publish/subscribe. In *DEBS*, 2012.

[2] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *OSDI '14*.

[3] R. Pires, M. Pasin, P. Felber, and C. Fetzer. Secure content-based routing using intel software guard extensions. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 10:1–10:10, New York, NY, USA, 2016. ACM.