Quantum Oblivious Key Distribution Emulator

Prof. Armando Pinto, Zeinab Rahmani

June 2020

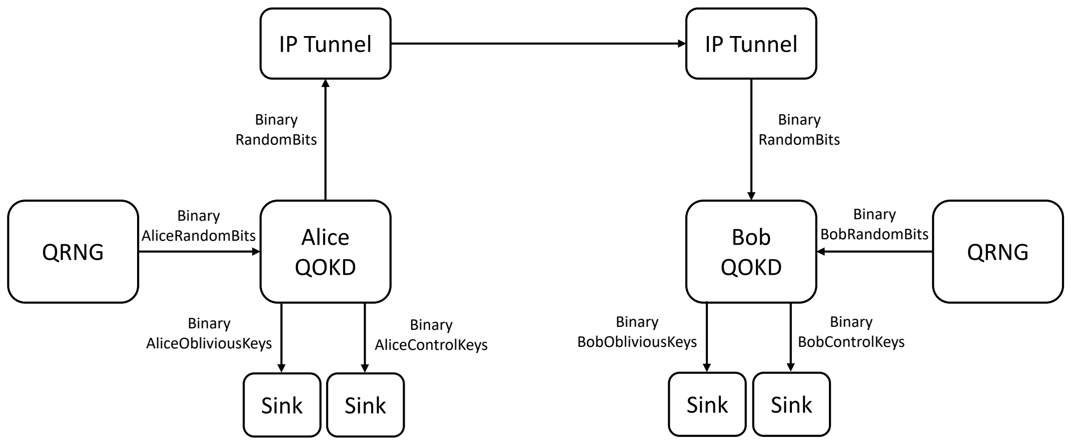


Figure 1: Quantum oblivious keys distribution emulator

1 Quantum Oblivious Key Distribution Emulator

In this section, we aim to introduce the Quantum Oblivious Key Distribution (QOKD) emulator. First we describe the structure of the emulator and its functionality. Afterwards, we explain how to use emulator in both MS Windows and Linux platforms. At the end, we demonstrate the output results and implementation steps in great details.

1.1 What Are Oblivious keys?

Oblivious keys are asymmetric keys, in which one of the parties (Alice) knows all the keys while the other (Bob) only know half of the keys.

1.2 What Is QOKD Emulator

Quantum Oblivious Key Distribution (QOKD) emulator consists of two programs that are implemented in the NetXpto framework. The programs are called "qokd_emulator_tx.exe" and "qokd_emulator_rx.exe" and are connected to each other using an IP link connection. The aim of the emulator is to generate and distribute the oblivious keys between parties (Alice and Bob) who may be in far or close distances.

1.3 What Are Requirements of the QOKD Emulator

The QOKD emulator has been developed in both MS Windows (x64 and x86) and Ubuntu 18.04.4 LTS (gcc 10.1.0) platforms. It is worth mentioning that, for the correct functionality of the emulator, C++17 is required.

1.4 How to Use QOKD Emulator?

The QOKD emulator is placed in the "quantum_oblivious_key_distribution" zip file in which both MS Windows and Linux versions of the emulator are implemented. For each version, there are two folders "tx" and "rx" each of which contains the "qokd_emulator_tx.exe" and "qokd_emulator_rx.exe" files corresponding to Alice and Bob, respectively.

Input Parameter File

In order to create a user-system interaction, an input parameter file has been prepared for each party. Using this file, user can make desired changes to input parameters such as iPAddress, numberOfSectors, etc. This can be accomplished using one of the following approaches:

- **Loading input parameters from a pre-defined file** To this end, the two text files "input_parameter_tx.txt" and "input_parameter_rx.txt" are prepared for each party. Using theses files, the user can manually set the desired input values such as IP Addresses for the specific machines.
- **Loading input parameters from a user-defined file** Using this approach, user is able to supply a new input parameter file to the program through the command line. To this end the following command should be executed in the command line:

  ```
  cd <path_to_some_system.exe>
  some_system.exe <input_file_name>
  ```

  where *some_system.exe* is the name of the executable generated after compiling the project, in this case "qokd_emulator_tx.exe" or "qokd_emulator_rx.exe" and <*input_file_name*> is the name of the new input parameters file.
  Note that for the case of user-defined file, it is necessary to respect the defined file format, otherwise the program wont be able to read the input parameters.

Input Variables

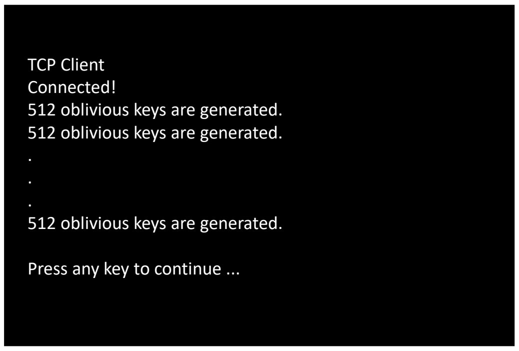
The input variables of the input parameter file are shown in table 1. Using the input parameter file, user is able to change the default values to the desired amounts.

System Execution in MS Windows

The emulator could be executed in one or two machines as follows:

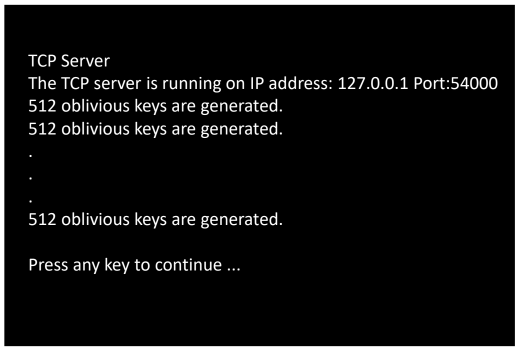
Input Parameter	Decryption	Default
remoteMachineIpAddress	IP address of the remote machine that IPTunnel will try to connect to	127.0.0.1
localMachineIpAddress	IP address of the local machine	127.0.0.1
numberOfSectors	Number of sectors (each sector contains 512 bytes of oblivious keys)	100

Table 1: Input variables



```
TCP Client
Connected!
512 oblivious keys are generated.
512 oblivious keys are generated.
.
.
.
512 oblivious keys are generated.

Press any key to continue ...
```



```
TCP Server
The TCP server is running on IP address: 127.0.0.1 Port:54000
512 oblivious keys are generated.
512 oblivious keys are generated.
.
.
.
512 oblivious keys are generated.

Press any key to continue ...
```

Figure 2: Dashboard consoles outputs

- **QOKD in one machine** The two folders "tx" and "rx" could be placed in any working directory. Afterwards, by simultaneous execution of the **.exe* files ("qokd_emulator_tx.exe" and "qokd_emulator_rx.exe"), two dashboard consoles appears indicating the correct performance of the system. Note that when the two executable files are placed in one machine, identical IP Addresses are applied, and it is not necessary to change the IP Address in the input parameter files. A pictorial representation of the output consoles is shown in Fig. 2. As it can be seen, the total number of generated keys are printed for each sector.

- **QOKD in two machines** In order to start the distribution process between two machines, the folders "tx" and "rx" should be placed in machine 1 and machine 2, respectively. Since the emulator uses TCP protocol to establish the connections between the two machines, it is necessary to specify the IP Addresses of remote and local computers for each party. This can be accomplished using the input parameter files.
  Despite the use of identical IP Addresses in one machine, for two machines, different IP Addresses should be used for machine 1 and machine 2. After defining the IP Addresses, similar to previous scenario, the two *.exe files should be executed simultaneously leading to appearance of the output consoles as shown in Fig 2. The final outputs of the emulator are stored in the "signal" folder that is created during the execution process in the working directory.

System Execution in Linux

Similar the previous scenario, the emulator could be executed in one or two machines. As mentioned before for the case of two machines, it is necessary to specify the IP Addresses of remote and local machines through the input parameter file that is prepared for each party. Before starting the process, it is necessary to update the system by executing the following commands in terminal:

```
sudo apt update
sudo apt upgrade
sudo apt install build-essential
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt update
sudo apt install gcc-6
sudo apt-get install libstdc++6
```

For the direct execution of the emulator in Linux platform the two following options are available:

- **Option 1: Using Linux build outputs** To this end, the two output files "qokd_emulator_tx" and "qokd_emulator_rx" are provided. To run the emulator, apply the following steps:

  1. Open the terminal and go to the `quantum_oblivious_key_distribution\linux\tx` directory and execute the following commands:

     ```
             chmode +x qokd_emulator_tx
             ./qokd_emulator_tx
     ```

  2. Open a new terminal and go to the `quantum_oblivious_key_distribution\linux\rx` directory and execute the following commands:

     ```
             chmode +x qokd_emulator_rx
             ./qokd_emulator_rx
     ```

  Note that the chmode +x command is to make sure that the build outputs ("qokd_emulator_tx" and "qokd_emulator_rx") are recognizable as executable files. By simultaneous execution of the output files, two dashboard consoles appears as shown in Fig. 2.

- **Option 2: Using MS Windows build outputs** Using this option, we can take the advantage of using the output **.exe* files generated in MS Windows directly.

However, since the binary formats for Linux and MS Windows are different, a middleware such as *"Wine"* is required to convert the binaries for Linux. The first step is to install *"Wine"*. Afterwards, execute the following command in terminal, in the corresponding working directories:

```
wine some_system.exe
```

By simultaneous executing of this command both for "qokd_emulator_tx.exe" and "qokd_emulator_rx.exe" in the corresponding working directories, the distribution process starts between the two parties. The next steps are exactly similar to what was explained for the MS Windows platform. Note that, although the execution of the emulator through *Wine Middleware* is straightforward, it cant be considered efficient for large complex applications.

1.5 What Are QOKD Outputs?

As mentioned before, the emulator is consists of two programs corresponding to Alice and Bob. As shown in Fig. 1, the program "qokd_emulator_tx.exe" sends out the following output signals:

- RandomBits (bit)
- AliceObliviousKeys (bit)
- AliceControlKeys (bit)

The program "qokd_emulator_rx.exe" sends out the following output signals:

- BobObliviousKeys (bit)
- BobControlKeys (bit)

Theses final outputs are stored in the "signal" folder that is created during the execution process, in both "tx" and "rx" folders (offline). Note that in addition to the output folders, the signals are stored in the sink (internal container) of the emulator to be used in real time for future purposes.

Important note about output signals

1. Every time you run the emulator, you receive different output keys (because the source of key generation performs randomly each time).
2. The output signals are binary values of type "unsigned int". In other words, the signals contain sequences of 0s and 1s which their type are defined as "unsigned int" in NetXpto platform.

3. All the output files contain four header lines in which the type of signal, as well as some additional information related to the signal features are recorded.

4. If you need to read from the output .sgn file, its necessary to open the file in binary format, in your C++ project.

5. While reading from file, header lines should be skipped through coding (do not removing them manually through a text editor) because if you remove the header lines manually, it will corrupt the data that is stored in the output files.

6. The minimum generated key length is 512, which means that at least 512 bits of oblivious keys will be generated, in addition to the amount that is specified in the input parameter file.

1.6 How to Interpret Output Signals through Matlab Simulator?

Since output signals of the QOKD emulator are binary signals, they can not be interpreted correctly through the ordinary text editor such as Notepad. Therefore, the Matlab script "readSignal.m" is prepared to represent binary values as sequences of 0s and 1s streams. The only necessary step is to pass the signal file name through the readSignal function as follows:

```
readSignal( fname );
```

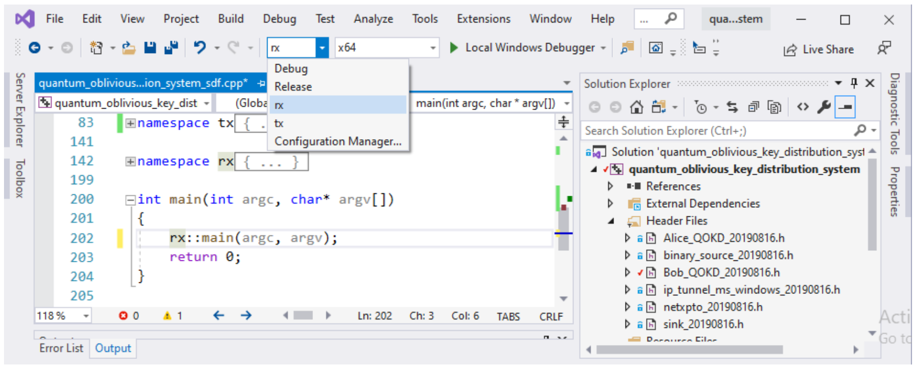
Note that the use of Matlab script is only to see the actual values of the .sgn files in printable format. However, you can read the file through a C++ project or GNU Octave, instead of using Matlab script.

1.7 How to Regenerate QOKD Build Outputs?

Although the output executable files of the QOKD emulator are already created, one may desire to regenerate the emulator from the scratch. To this end, the folder "source_codes" is placed in the emulator folder. To regenerate the outputs, three following options are provided:

- **Compile QOKD in Visual Studio** To execute the emulator in VS, the following steps should be applied:

    1. Go to the `quantum_oblivious_key_distribution\source_codes\visual_studio`.
    2. Open the "quantum_oblivious_key_distribution.sln" file in Visual Studio IDE.


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
qua...stem
rx
x64
Local Windows Debugger
Live Share
Debug
Release
rx
tx
Configuration Manager...
quantum_oblivious...ion_system_sdf.cpp*
quantum_oblivious_key_dist
main(int argc, char * argv[])
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'quantum_oblivious_key_distribution_syst
quantum_oblivious_key_distribution_system
References
External Dependencies
Header Files
Alice_QOKD_20190816.h
binary_source_20190816.h
Bob_QOKD_20190816.h
ip_tunnel_ms_windows_20190816.h
netxpto_20190816.h
sink_20190816.h
118 %
Ln: 202 Ch: 3 Col: 6 TABS CRLF
Error List Output
Server Explorer Toolbox
Diagnostic Tools Properties

```
namespace tx { ... }
namespace rx { ... }

int main(int argc, char* argv[])
{
    rx::main(argc, argv);
    return 0;
}
```

Figure 3: Visual studio build setting for "rx"

3. Open the "quantum_oblivious_key_distribution_sdf" file and build the project in two separate steps: one for "tx" and the other for "rx". Note that in each step the corresponding configuration manager, as well as the correct main function "tx::main(argc, argv);" or "rx::main(argc, argv);" should be set. For instance the correct setting for the "rx" is shown in Fig. 3.
4. After finalizing the build process, the "tx" and "rx" folders are created each of which contains the build outputs corresponding to Alice and Bob, respectively.
5. Afterwards, its necessary to copy the corresponding input parameter files ("input_parameter_tx.txt" and "input_parameter_rx.txt") in the "tx" and "rx" folders, respectively.
6. Finally by simultaneous execution of the **.exe* files, the distribution process starts as shown in Fig. 2.

- **Compile QOKD through Makefile** To regenerate the build outputs through Terminal in Ubuntu, the following steps should be applied:
  1. Open the terminal and go to the `quantum_oblivious_key_distribution/source_codes/terminal/tx` and type `make`. After running the make command, the build output "qokd_emulator_tx" should be created in the "tx" folder.
  2. Open a new terminal and go to the `quantum_oblivious_key_distribution/source_codes/terminal/rx` and type `make`. Afterwards, the build output "qokd_emulator_rx" should be created in the "rx" folder.
  3. After creation of the build outputs ("qokd_emulator_tx" and "qokd_emulator_rx"), we are able to execute the emulator as follows:

- Open the terminal and go to the `quantum_oblivious_key_distribution\source_codes\makefile\tx` and type `./qokd_emulator_tx`
- Open a new terminal and go to the `quantum_oblivious_key_distribution\source_codes\makefile\rx` and type `./qokd_emulator_rx`

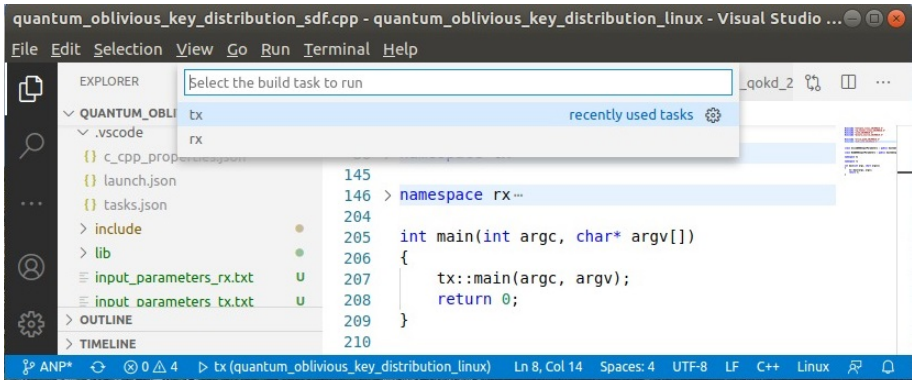
By simultaneous execution of the output files, two dashboard consoles appears as shown in Fig. 2.

- **Compile QOKD in Visual Studio Code** As explained before, the QOKD Emulator is consists of two programs that are connected to each other through an IP Link called IP Tunnel. However, at the current implementation of NetXpto, the IP Tunnel was only developed for MS Windows platform. Therefore, a new version of IP Tunnel has been developed enabling the direct execution of the emulator in Linux platform. A detailed explanation of IP Tunnel implementation has been prepared in "ip_tunnel_linux_2019816" library section of the linkplanner_smc repository. To run the emulator, the "visual_studio_code" folder is prepared in the "source_cods" directory. Note that the VS Code is chosen only for the sake of similarity to the Visual Studio for MS Windows. However, the emulator could be executed through terminal or any other IDE. To execute the emulator in VS Code, the following steps should be applied:

  1. Go to the `quantum_oblivious_key_distribution\source_codes\visual_studio_code`.
  2. Open the "visual_studio_code" folder through VS Code.
  3. Go to the "quantum_oblivious_key_distribution_sdf" file and build the task both for "tx" and "rx" as follow (for the sake of simplicity only "tx" is explained here):
     - Set the namespace as:

```
int main(int argc, char* argv[]){
  tx::main(argc, argv); //for "rx" it
     should be rx::main(argc, argv);
  return 0;
}
```

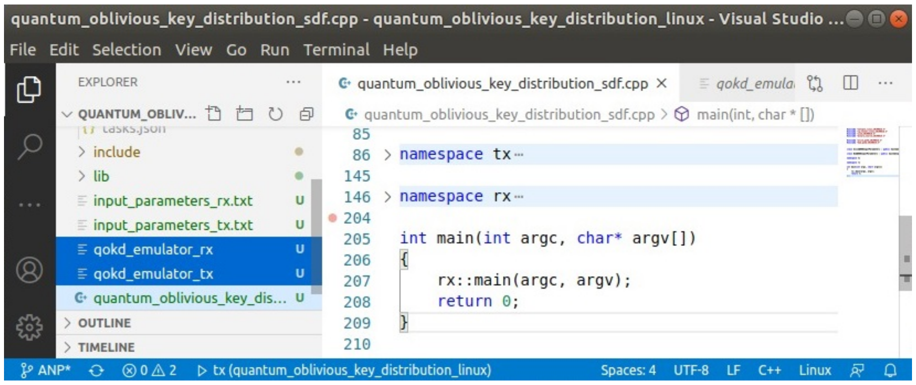
     - In the menu bar -> **Terminal** -> **Run Build Task** (ctrl+shift+B)
     - Following the previous step, a windows appears representing the existing configurations as shown in Fig.4, in this case "tx" and "rx". By pressing the "tx" (or "rx" in the case of receiver), the build process is started leading to creation of "qokd_emulator_tx" output


quantum_oblivious_key_distribution_sdf.cpp - quantum_oblivious_key_distribution_linux - Visual Studio ...
File Edit Selection View Go Run Terminal Help
EXPLORER
Select the build task to run
tx
recently used tasks
rx
QUANTUM_OBLI
.vscode
c_cpp_properties.json
launch.json
tasks.json
include
lib
input_parameters_rx.txt
input_parameters_tx.txt
OUTLINE
TIMELINE
ANP* ⊗0 ⚠4 ▷ tx (quantum_oblivious_key_distribution_linux) Ln 8, Col 14 Spaces: 4 UTF-8 LF C++ Linux

```
namespace rx…

int main(int argc, char* argv[])
{
    tx::main(argc, argv);
    return 0;
}
```

Figure 4: VS Code build setting for "tx"


quantum_oblivious_key_distribution_sdf.cpp - quantum_oblivious_key_distribution_linux - Visual Studio ...
File Edit Selection View Go Run Terminal Help
EXPLORER
quantum_oblivious_key_distribution_sdf.cpp
qokd_emula
QUANTUM_OBLIV...
quantum_oblivious_key_distribution_sdf.cpp > main(int, char * [])
tasks.json
include
lib
input_parameters_rx.txt
input_parameters_tx.txt
qokd_emulator_rx
qokd_emulator_tx
quantum_oblivious_key_dis...
OUTLINE
TIMELINE
ANP* ⊗0 ⚠2 ▷ tx (quantum_oblivious_key_distribution_linux) Spaces: 4 UTF-8 LF C++ Linux

```
namespace tx…

namespace rx…

int main(int argc, char* argv[])
{
    rx::main(argc, argv);
    return 0;
}
```

Figure 5: VS Code build outputs both for "tx" and "rx"

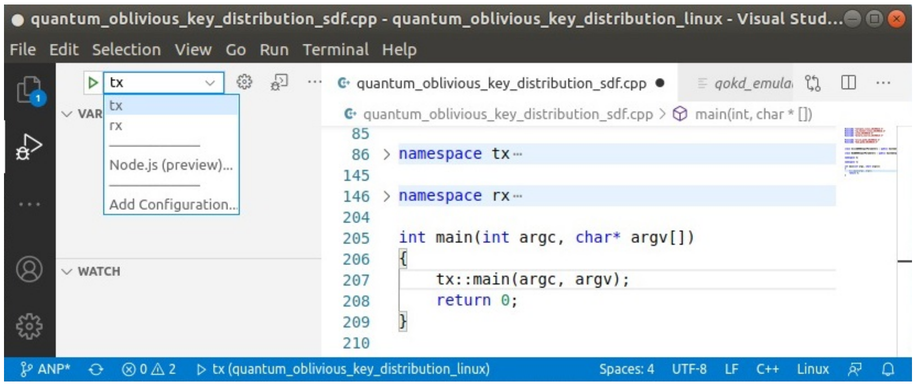
file. By repeating the exact same steps for "rx", the output build "qokd_emulator_rx" is also created as highlighted in Fig 5.

4. After building the task both for "tx" and "rx", we are able to run the emulator as shown in Fig. 6. finally, the two output console appear indicating the correct performance of the emulator as shown in Fig. 2.

1.8 Implementation of QOKD Emulator

In this section we explain the implementation of the Quantum Oblivious Key Distribution (QOKD) emulator, in which a sequence of asymmetric keys are generated and distributed among parties. The first step towards implementing the QOKD emulator is the generation of random bits through Quantum Random Number Generator (QRNG) block for Alice. The AliceQOKD block receives the signal and sends out the following output signals:

- RandomBits: Equal to the bit stream generated by QRNG block.


quantum_oblivious_key_distribution_sdf.cpp - quantum_oblivious_key_distribution_linux - Visual Stud...
File Edit Selection View Go Run Terminal Help
tx
tx
rx
Node.js (preview)...
Add Configuration...
VAR
WATCH
quantum_oblivious_key_distribution_sdf.cpp
qokd_emula
quantum_oblivious_key_distribution_sdf.cpp > main(int, char * [])
ANP* ⊗0 ⚠2 ▷ tx (quantum_oblivious_key_distribution_linux)
Spaces: 4 UTF-8 LF C++ Linux

```cpp

> namespace tx···

> namespace rx···

int main(int argc, char* argv[])
{
    tx::main(argc, argv);
    return 0;
}

```

Figure 6: Run the QOKD emulator through VS Code both for "tx" and "rx"

- AliceObliviousKeys (bit): This signal carries the Alice's oblivious keys. For instance: 0, 1, 1, 0, 1, 0, 0, ...

- AliceControlKeys (bit): always bit 0: $0, 0, 0, \ldots$. This signal is used to tell the parties how many bits of the oblivious keys are revealed to them. This is done in a way that the revealed bits are shown with 0, and the hidden bits with 1. In this case, since Alice knows all keys, the signal is only sequences of 0s.

The first random bit stream is send to BobOKD block through a communication channel called IP Tunnel. The BobQKD receives the signal and substitute one element of each pair with a new random bit that has been generated using second QRNG block. This substitution leads to generation of oblivious keys. In addition to the Bob oblivious keys, an control signal will be generated to represent the correlated and uncorrelated bits, for Bob as follows:

- BobObliviousKeys (bit): equal to the received Signal RandomBits, with exception that the one element of each pair is replaced by a random generated bit, in the present case it would be: $0, R, R, 0, 1, R, 0, R, \ldots$.

- BobControlKeys (bit): This signal tells Bob which bits are perfectly correlated with Alice's bits and which are not. For instance: $0, 1, 1, 0, 0, 1, 0, 1, \ldots$.

It is worth mentioning that, after generation of the output signals, they are put in the sink container to be used for the future purposes. Note that neither Alice nor Bob has access to the performed operations in the QOKD blocks. This confidentiality is required for the proper performance of the distribution process, so that the Alice does not know which bits of the evaluator keys are correlated with her keys and Bob knows only half of the keys.