



Installation, configuration & administration.

Post-Quantum Cryptography (PQC)

# User manual

version 1.0 Beta

August 2020.

MiniPCQ is a registered trademark of OpenQbit Inc., under a free use and commercial license. Terms and conditions of use at: [www.OpenQbit.com](http://www.OpenQbit.com)

## Content

1. Introduction. ....	3
2. What is Blockly programming? .....	4
3. What is Termux? .....	4
4. What is Mini PQC?.....	4
5. Storage configuration within Termux. ....	8
6. SSH (Secure Shell) server installation.....	9
7. SSH server configuration on mobile phone (smartphone). ....	10
8. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable). ....	25
9. Definition and use of blocks in Mini PQC.....	26
10. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable). ....	38
11. Annex "Python programs".....	39
12. Annex "OpenQbit Quantum Computing".....	50
13. Licensing and use of software. ....	54

## 1. Introduction.

Historically speaking, cryptography can be classified into 4 stages: classical, modern, quantum and post-quantum. The first recognized use of cryptography dates to about 4000 years ago, in some non-standard hieroglyphs found in an Egyptian tomb. Classical cryptography, in one form or another, continued to be used as human civilization continued to evolve.

The Second World War marks an important milestone for cryptography. In view of the need to protect messages sent to troops, planes and submarines from a distance, great progress was made in both the construction of cryptographic machines.

The next great revolution appeared in the late 1960s with the development of asymmetric cryptography where a publicly known key is used to encrypt messages and another private key to decrypt them.

In the 70's the first ideas related to quantum cryptography were had, highlighting Shor and Grover's algorithms. However, it was not until the 80's that the first publications of new protocols were shown that based their security on the principles of quantum mechanics - such as uncertainty or superposition- using lasers to emit information on a photon.

When the algorithms of public-key cryptography begin to be compromised by quantum computation, post-quantum cryptography emerges, which refers to algorithms designed to resist quantum computer attacks. There are several branches that differ in the way they operate, being these: grid-based algorithms, algorithms based on multivariate equations, algorithms based on supersingular isogenic elliptic curves, hash-based algorithms, and code-based algorithms.

From the point of view of the use and performance of this type of algorithms, its execution in systems with ARM architecture basically oriented to mobile phones as they have been explored: Android, Arduino and Raspberry Pi. Hence, the background related to this work exhibits works with post-quantum algorithms that could be divided into executions in ARM based devices and Android operating system, in specific use devices and post-quantum algorithm works in ARM architecture, which lie in encryption, digital signatures and key agreements, leaving open the possibility of exploring the hash primitive, algorithms such as **CHACHA20**, **NTRU**, **NEWHOPE**, **SALSA20**, **SPHINCS+**, **RAIBOW** and **CODECRYPT**, focused on integrity service, as well as its behavior in systems with limited resources.

## 2. What is Blockly programming?

**Blockly** is a **visual programming language** composed of a simple set of commands that we can combine as if they were the pieces of a puzzle. It is a very useful tool for those who want to **learn how to program** in an intuitive and simple way or for those who already know how to program and want to see the potential of this type of programming.

Blockly is a form of programming where you don't need any background in any kind of computer language, this is because it is just joining graphic blocks as if we were playing lego or a puzzle, you just need to have some logic and that's it!

Anyone can create programs for mobile phones (smartphones) without messing with those programming languages difficult to understand, just put together blocks in a graphical way in a simple, easy and fast way to create.

## 3. What is Termux?

Termux is an Android terminal emulator and a Linux environment application that works directly without the need for rooting or configuration. A minimal base system is automatically installed.

We will use Termux for its stability and easy installation and management, however, you can use an installed environment of Ubuntu Linux for Android.

In this Linux environment you will have the "core" of the MiniPQC Cryptographic Post Quantum algorithm processes.

## 4. What is Mini PQC?

Mini PQC (Mini Post-Quantum Cryptography) is Software that includes the following technological solutions (algorithms) to be able to create PQC (Post-Quantum Cryptography) as follows

- CHACHA20
- NEWHOPE
- NTRU
- SALS20
- SPHINCS+
- RABOW
- CODECRYPT

## 1. Installation and configuration of Termux Terminal.

First we need a Linux environment since every Android system is based on Linux for security and flexibility in tools, we will use the "Termux" terminal that contains that environment where we will install the tool(s) that will help us to create QRNGs.

Termux is a Linux emulator where we will install the necessary packages to create quantum numbers.

One of the main advantages of using Termux is that you can install programs without having to "rotate" the mobile phone (Smartphone). This ensures that no manufacturer's warranty is lost due to this installation.

Termux installation.

From your mobile, go to the Google Play icon application ([play.google.com](https://play.google.com)).



Search by application "Termux", select it and start the installation process.



## Start of the Termux application.

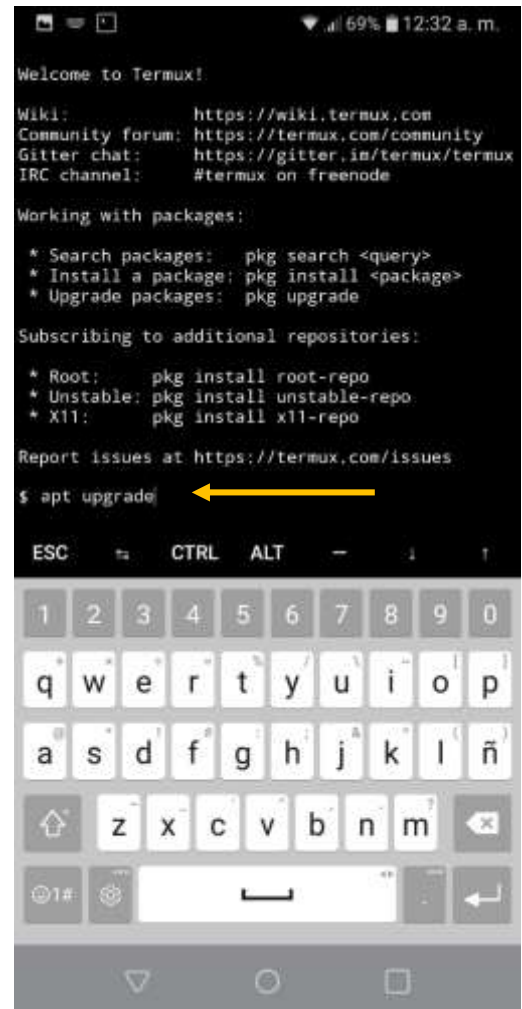
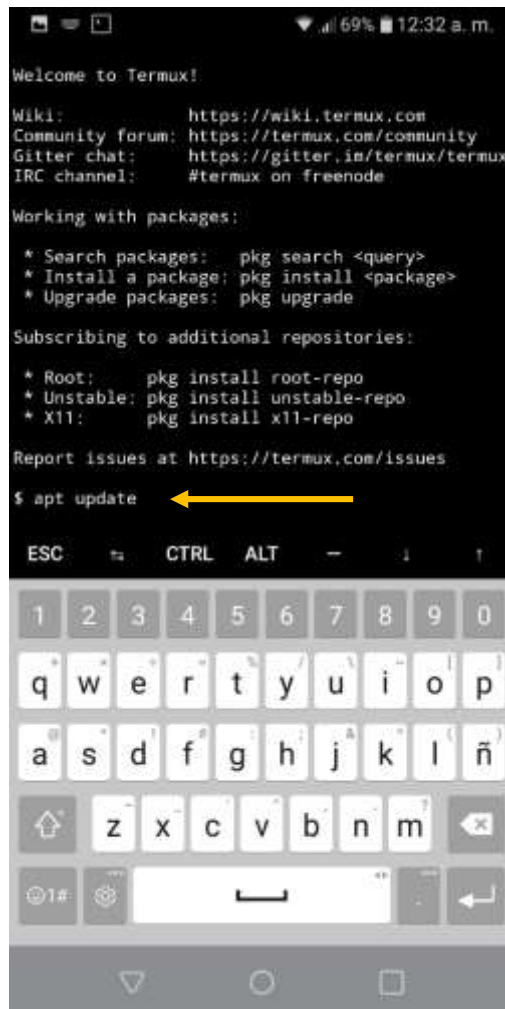
After starting we will have to execute the following two commands to perform updates of the Linux operating system emulator:

\$ apt update

\$ apt upgrade

Confirm all options Y(Yes)...

Termux Home \$ apt update \$ apt upgrade



## 5. Storage configuration within Termux.

After you have updated and upgraded the Termux system, we will start configuring how to view the internal storage of the phone in the Termux system. This will help you to be able to exchange information between Termux and our information in the phone.

This can be done simply and quickly by running the following command on a Termux terminal.

**\$ termux-setup-storage**

When you execute the previous command, a window appears asking you to confirm the creation of a virtual **storage** (directory) in Termux. We verify by giving the command:

**\$ ls**





## 6. SSH (Secure Shell) server installation.

```
$ apt install openssh
```

```
$ apt install sshpass
```

```
$ apt install openssh $
```

```
apt install sshpass
```



```
$ apt install openssh
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  krb5 libdns libdb libedit termux-auth
The following NEW packages will be installed:
  krb5 libdns libdb libedit openssh termux-auth
0 upgraded, 6 newly installed, 0 to remove and 0
not upgraded.
Need to get 2255 kB of archives.
After this operation, 11.9 MB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm libdb arm 18.1.32-4 [465
kB]
Get:2 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm krb5 arm 1.18.1 [839 kB]
24% [2 krb5 131 kB/839 kB 16%]
```



```
$ apt install sshpass
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  sshpass
0 upgraded, 1 newly installed, 0 to remove and 0
not upgraded.
Need to get 7158 B of archives.
After this operation, 57.3 kB of additional disk
space will be used.
0% [Working]
```

We have finished with the installation of the communication network for localhost SSH server on mobile Smartphone.

## 7. SSH server configuration on mobile phone (smartphone).

We will enable the SSH server in the mobile phone to be able to connect from our PC to the mobile phone and to be able to work in a faster and more comfortable way, also it will serve us to check that the service of the SSH server in the mobile phone works correctly since we will use it in the communication with Mini QRNG.

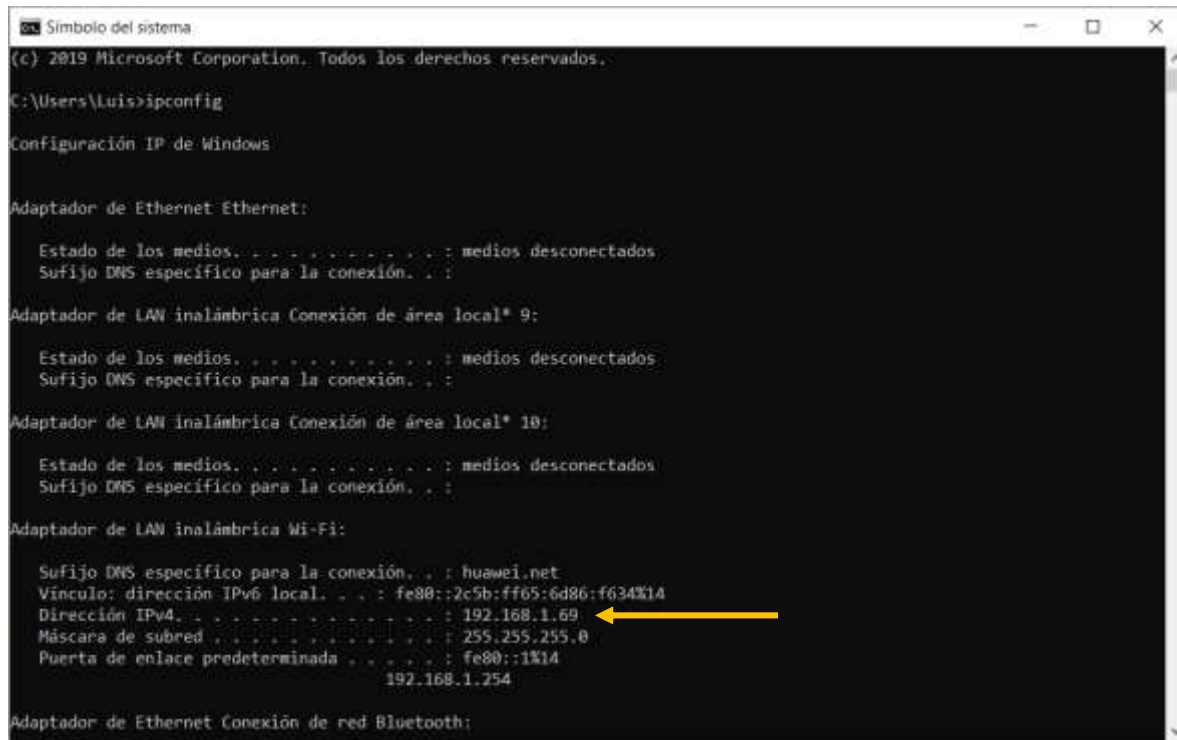
The first thing we have to do is connect the mobile and the PC to the **same WiFi network** so that they can see each other. The IPs or addresses must be similar to 192.168.XXX.XXX the XXX values are variable numbers that are assigned randomly in each computer.

This example was tested on an LG Q6 mobile phone and a PC with Windows 10 Home.

Check the IP or address that the PC has connected to the WiFi we must open a terminal in Windows.

In the bottom panel where the search magnifier is write cmd and press the Enter key. A terminal will open and in it we write the command:

```
C:\User_Name> ipconfig
```



```
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Conexión de área local* 9:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Conexión de área local* 10:

    Estado de los medios. . . . . : medios desconectados
    Sufijo DNS específico para la conexión. . :

Adaptador de LAN inalámbrica Wi-Fi:

    Sufijo DNS específico para la conexión. . : huawei.net
    Vínculo: dirección IPv6 local. . . : fe80::2c5b:ff65:6d86:f634%14
    Dirección IPv4. . . . . : 192.168.1.69
    Máscara de subred. . . . . : 255.255.255.0
    Puerta de enlace predeterminada. . . : fe80::1%14
                                      192.168.1.254

Adaptador de Ethernet Conexión de red Bluetooth:
```

It will show us the IP assigned to the PC in this is 192.168.1.69 but this is most likely to be different in each case.

NOTE: The address where it says "IPv4 address" should be taken, not to be confused with the Gateway.

Now in the case of the mobile phone in the Termux terminal we must type the following command to know the name of our user that we will use to connect to the SSH server that has our phone, we execute the following command:

**\$ whoami**

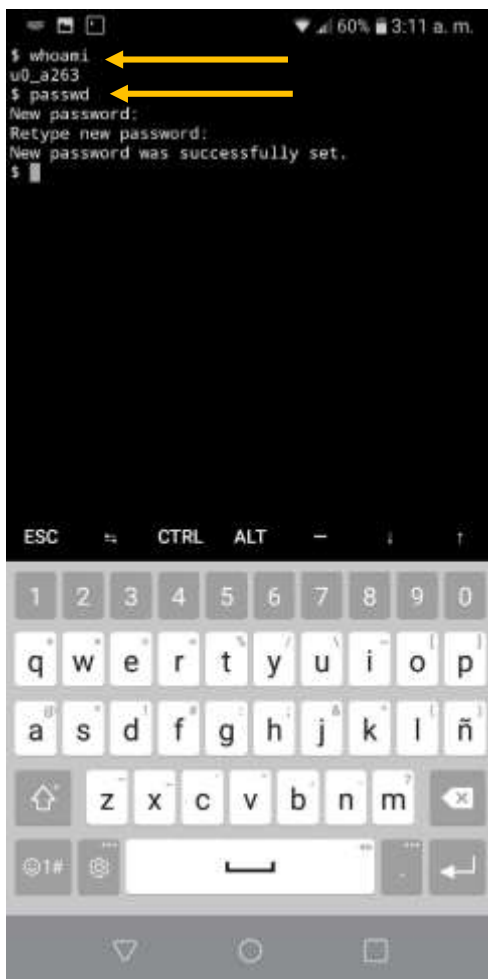
Later we must give a password to this user so we have to execute the following command:

**\$ passwd**

It will ask us to type a password and hit Enter, again it asks us for the password we confirm it and hit Enter, if it has been successfully **"New password was successfully set"** in case of marking an error is possible that the password has not been typed correctly. Perform the procedure again.

And then to know what IP we have in Termux we type the following command, the IP is after the word **"inet"**:

**\$ ifconfig -a**



The screenshot shows a Termux terminal window on a mobile phone. The status bar at the top indicates 60% battery and 3:11 a.m. The terminal displays the following commands and output:

```
$ whoami
u0_a263
$ passwd
New password:
Retype new password:
New password was successfully set.
$
```

Two yellow arrows point to the 'u0\_a263' output of the 'whoami' command and the 'passwd' command prompt.



The screenshot shows a Termux terminal window on a mobile phone. The status bar at the top indicates 61% battery and 2:57 a.m. The terminal displays the output of the 'ifconfig -a' command:

```
e 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carri
er 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST
>  mtu 1500
    inet 192.168.1.68  netmask 255.255.255.0
    broadcast 192.168.1.255
    inet6 fe80::257:c1ff:fee6:3051  prefixle
n 64  scopeid 0x20<link>
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
    txqueuelen 1000  (UNSPEC)
    RX packets 908745  bytes 947916536 (904.
0 MiB)
    RX errors 0  dropped 0  overruns 0  fram
e 0
    TX packets 601034  bytes 93496881 (89.1
MiB)
    TX errors 0  dropped 0 overruns 0  carri
er 0  collisions 0

$
```

A yellow arrow points to the 'inet 192.168.1.68' line, indicating the IP address.

Now it's time to start the SSH server service on your phone so you can receive sessions from your PC. We execute the following command in the Termux terminal, this command does not give any result.

\$ sshd



Now we will have to install a program on the PC that will communicate with the phone's SSH server from the PC.

We have to go to <https://www.putty.org>

Select where the link "You can download PuTTY here" is



### Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows operating system, with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).

---

Below suggestions are independent of the authors of PuTTY. They are *not* to be seen as

---



### Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported professionally and supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

Choose the 32-bit version, it doesn't matter if your system is 64-bit will work fine.

**Download PuTTY: latest release**

[Home](#) | [FAQ](#) | [Feedback](#) | [Licence](#) | [Updates](#) | [Mirror](#)  
Download: [Stable](#) · [Snapshot](#) | [Docs](#) | [Contact](#)

This page contains download links for the latest released version of PuTTY. Currently this is 0.73, released on 2019-09-29.

When new releases come out, this page will update to contain the latest, so this is a good page to bookmark or link to. Alternative Release versions of PuTTY are versions we think are reasonably likely to work well. However, they are often not the most up-to-date or the [development snapshots](#), to see if the problem has already been fixed in those versions.

### Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

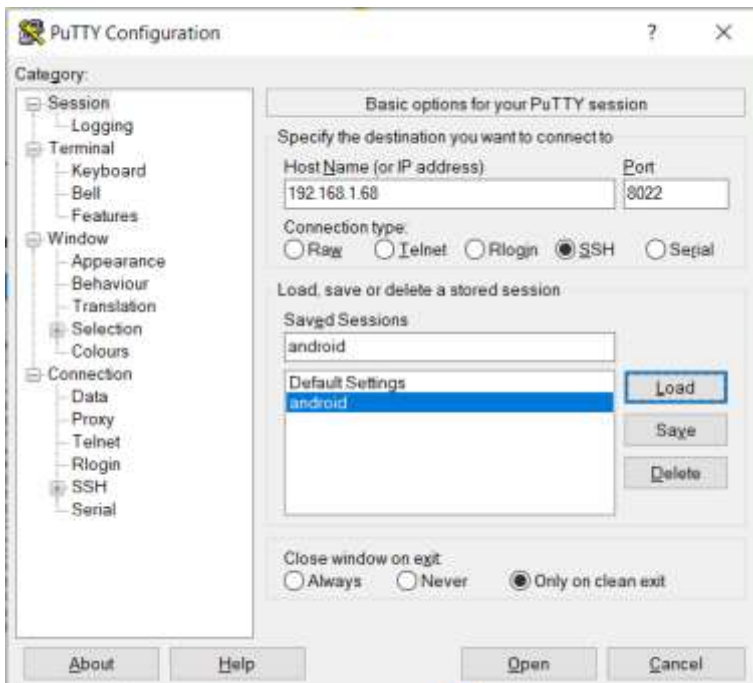
**MSI ('Windows Installer')**

32-bit:	<a href="#">putty-0.73-installer.msi</a>	(or by FTP)	(signature)
64-bit:	<a href="#">putty-64bit-0.73-installer.msi</a>	(or by FTP)	(signature)

**Unix source archive**

.tar.gz:	<a href="#">putty-0.73.tar.gz</a>	(or by FTP)	(signature)
----------	-----------------------------------	-------------	-------------

Once it has been downloaded to your PC, run it and install it with the default options. Then start the PuTTY application.



In this session we will enter the data from our Openssh server that we installed in the mobile phone.

Enter the IP of the mobile phone.

HostName or IP address:

**192.168.1.68** (IP example)

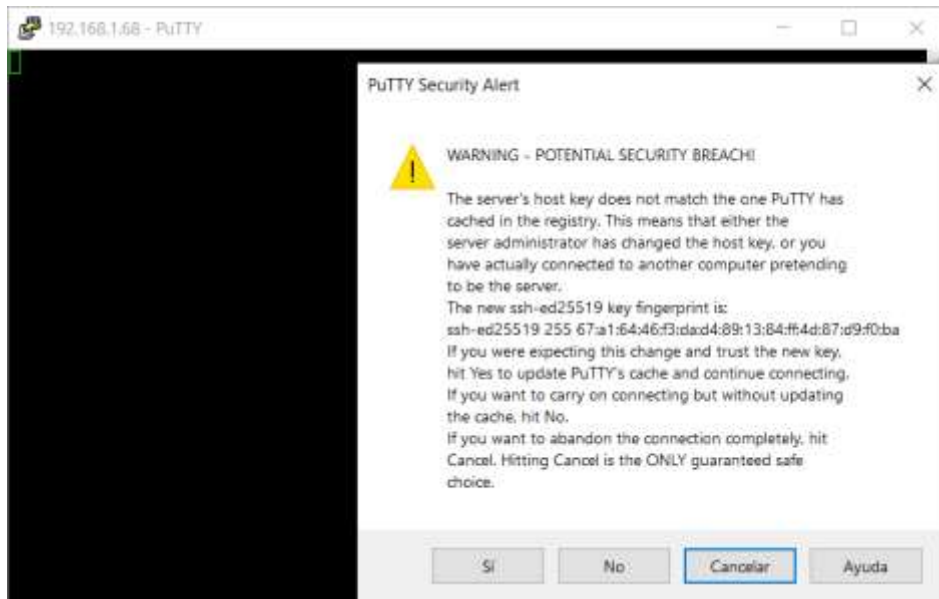
Port:

**8022** (Default port of the mobile SSH server).

We can give a name to the session in "Saved Sessions" and click on the Save button.

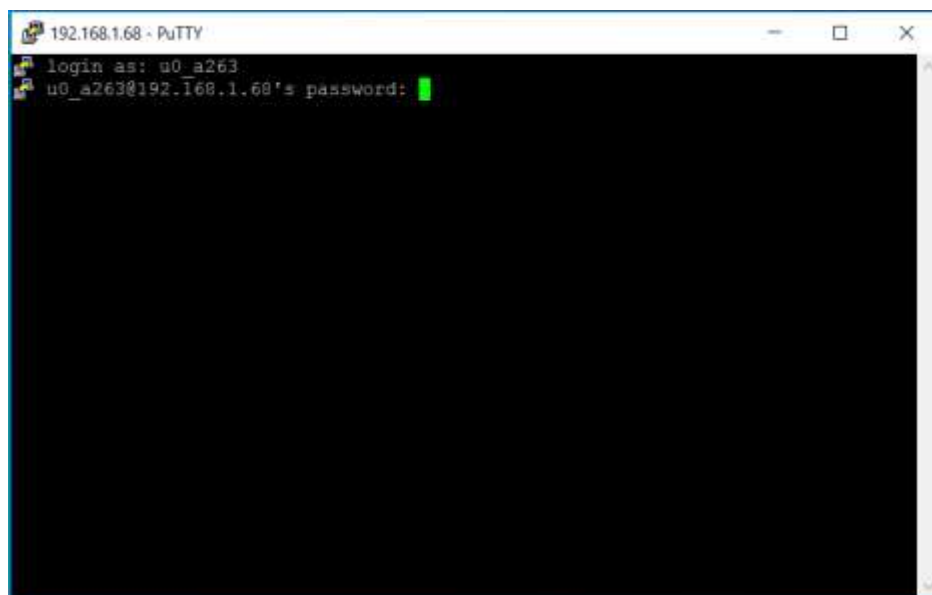
Later on in the lower part we press to open a connection to the server giving the button "Open".

On the PC when you connect for the first time you **will be asked for confirmation of the information encryption key** by clicking on the "Yes" button.



Later on, we will be asked for the user we are going to connect with. We will use the information that we previously obtained (user and password).

In the **Login as:** we must enter our user and give Enter, then we will ask for the password again give the Enter button.





Numpy module installation with the following command:

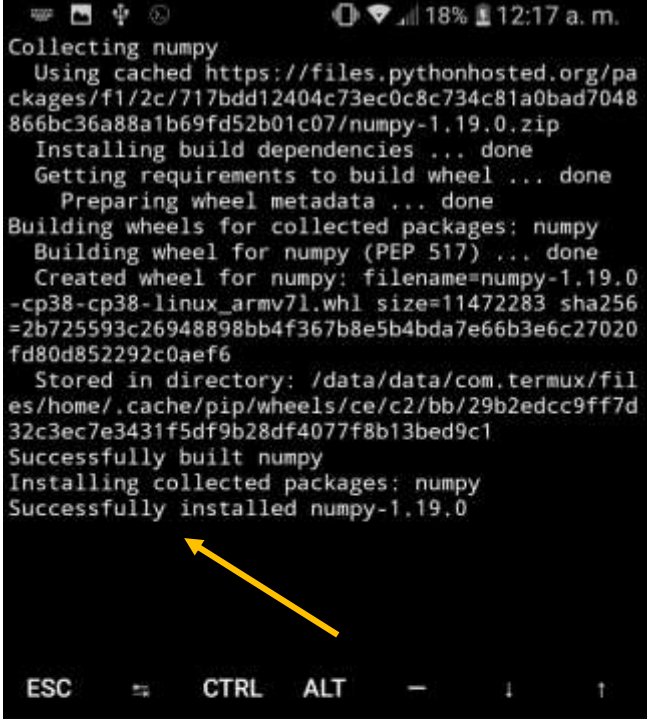
`apt install numpy` (Patience may take 40 minutes or more to install `numpy`)

`$ apt install git`



```

$ pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/f1/2c/717bdd12404c73ec0c8c734c81a0bad7048866bc36a88a1b69fd52b01c07/numpy-1.19.0.zip (7.3MB)
  10kB 1.0
  20kB 466
  30kB 561
  40kB 583
  51kB 513
  61kB 529
  71kB 600
  81kB 658
  92kB 704
  102kB 72
  112kB 72
  122kB 72
  133kB 72
  143kB 72
  153kB 72
  163kB 72
  174kB 72
  
```



```

Collecting numpy
  Using cached https://files.pythonhosted.org/packages/f1/2c/717bdd12404c73ec0c8c734c81a0bad7048866bc36a88a1b69fd52b01c07/numpy-1.19.0.zip
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing wheel metadata ... done
Building wheels for collected packages: numpy
  Building wheel for numpy (PEP 517) ... done
  Created wheel for numpy: filename=numpy-1.19.0-cp38-cp38-linux_armv7l.whl size=11472283 sha256=2b725593c26948898bb4f367b8e5b4bda7e66b3e6c27020fd80d852292c0aef6
  Stored in directory: /data/data/com.termux/files/home/.cache/pip/wheels/ce/c2/bb/29b2edcc9ff7d32c3ec7e3431f5df9b28df4077f8b13bed9c1
Successfully built numpy
Installing collected packages: numpy
Successfully installed numpy-1.19.0
  
```

Let's start the installation of the PQC (Post-Quantum Cryptography) algorithms.

Chacha20-POLY1305 algorithm implementation, is a cryptographic message authentication code created by Daniel J. Bernstein. It can be used to verify data integrity and message authenticity. A variant of Bernstein's Poly1305 that does not require AES has been standardized by the Internet Engineering Task Force in RFC 8439.

Installation of Chacha20-POLY1305 with the following command:

**\$ apt install chacha20poly1305**



```
$ pip install chacha20poly1305
Collecting chacha20poly1305
  Downloading chacha20poly1305-0.0.3.tar.gz (5.1
  kB)
Building wheels for collected packages: chacha20
poly1305
  Building wheel for chacha20poly1305 (setup.py)
  ... done
  Created wheel for chacha20poly1305: filename=c
hacha20poly1305-0.0.3-py2.py3-none-any.whl size=
6682 sha256=a08824a3c9b1c3e1aef708f197793a60ef90
5c477820e7442f94e2532af35b81
  Stored in directory: /data/data/com.termux/fil
es/home/.cache/pip/wheels/5c/a3/a0/0a27c3ae2fec0
a4c94bac9685bcb83735d69fd08fbc837433d
Successfully built chacha20poly1305
Installing collected packages: chacha20poly1305
Successfully installed chacha20poly1305-0.0.3
$
```

More information about chacha20-Poly1305:

<https://tools.ietf.org/html/draft-agi-tls-chacha20poly1305-01>



Implementation of PQC **NTRU** algorithm. This method accepts text files with short messages not exceeding 1-5 KBytes in size and can take a long time to encrypt and decrypt files.

ntru is a simple implementation of the NTRUEncrypt encryption system.

Polynomial operations are implemented using the SymPy library. It was conducted at the Faculty of Mathematics and Information Sciences of Warsaw University of Technology.

The NTRU public-key cryptography system represents a significant improvement in the world of public-key cryptography: stronger and smaller than virtually any other system in use and resistant to quantum computing.

`pip3 install sympy numpy docopt` (numpy was already installed in previous pages)

`$ git clone https://github.com/jkrauze/ntru.git`

```
$ pip3 install sympy
Collecting sympy
  Downloading sympy-1.6.1-py3-none-any.whl (5.8 MB)
    | 10 kB 3.
    | 20 kB 22
    | 30 kB 27
    | 40 kB 25
    | 51 kB 30
    | 61 kB 32
    | 71 kB 35
    | 81 kB 38
    | 92 kB 41
    | 102 kB 4
    | 112 kB 4
    | 122 kB 4
    | 133 kB 4
    | 143 kB 4
    | 153 kB 4
    | 163 kB 4
    | 174 kB 4
    | 184 kB 4
    | 194 kB 4
ESC  CTRL  ALT  -  _  |  !
```

```
$ pip3 install docopt
Collecting docopt
  Downloading docopt-0.6.2.tar.gz (25 kB)
Using legacy setup.py install for docopt, since package 'wheel' is not installed.
Installing collected packages: docopt
  Running setup.py install for docopt ... done
Successfully installed docopt-0.6.2
$
```

```
$ git clone http://github.com/jkrauze/ntru.git
Cloning into 'ntru'...
warning: redirecting to https://github.com/jkrau
ze/ntru.git/
remote: Enumerating objects: 87, done.
remote: Total 87 (delta 0), reused 0 (delta 0),
pack-reused 87
Unpacking objects: 100% (87/87), 19.22 KiB | 149
.00 KiB/s, done.
$
```

We then generate the key pair to use for encrypting and decrypting messages.

Create private and public keys (keypair) with the following command in the Termux terminal.

`$ cd ntru`

```
$ ./ntru.py -v gen 167 3 128 myKey.priv myKey.pub
```

```

$ ./ntru.py -v gen 167 3 128 myKey.priv myKey.pub
NTRU(N=167,p=3,q=128) initiated
g: Poly(-x**154 - x**141 - x**138 + x**134 - x**130 + x**111 - x**101 + x**96 - x**81 + x**77 + x**65 + x**64 - x**57 + x**52 - x**49 + x**47 + x**40 + x**22 - x**20 - x**19 + x**16 - 1, x, domain='ZZ')
g coeffs: Counter({-1: 11, 1: 11})
f: Poly(-x**166 + x**161 + x**160 + x**159 - x**158 - x**156 - x**155 + x**153 + x**152 - x**151 - x**150 + x**149 + x**148 + x**147 - x**146 + x**144 - x**143 + x**142 + x**140 + x**136 - x**133 - x**131 + x**130 - x**126 + x**125 + x**124 - x**123 + x**122 - x**121 + x**120 - x**119 - x**118 - x**117 - x**116 + x**110 + x**107 + x**104 - x**103 + x**102 + x**101 + x**100 + x**99 + x**98 - x**97 - x**96 - x**94 - x**93 - x**92 + x**91 + x**89 + x**88 + x**87 - x**85 - x**84 - x**82 + x**81 + x**80 - x**79 - x**77 + x**74 + x**73 + x**72 - x**71 - x**70 + x**68 - x**66 - x**65 - x**64 - x**63 - x**62 + x**61 - x**58 - x**57 + x**56 - x**55 + x**54 + x**52 - x**51 + x**49 + x**48 - x**47 + x**45 - x**44 - x**41 + x**40 + x**39 + x**38 + x**37 + x**34 - x**32 - x**28 - x**27 + x**25 + x**24 + x**23 - x**21 - x**20 + x**19 - x**18 - x**16 + x**15 + x**14 - x**13 - x**10 - x**9 - x**8 - x**3 + x**2 + 1, x, domain='ZZ')
f coeffs: Counter({1: 55, -1: 54})
f_p: Poly(-x**166 + x**164 - x**163 + x**162 - x**161 + x**160 - x**159 + x**156 - x**154 + x**153 - x**152 - x**151 - x**150 - x**148 - x**147 - x**146 + x**144 - x**143 + x**139 + x**138 + x**136 - x**135 - x**134 - x**132 + x**131 - x**130 + x**129 + x**127 + x**126 - x**124 - x**121 - x**120 + x**119 + x**117 + x**115 + x**114 - x**112 + x**111 + x**109 - x**108 - x**106 + x**105 + x**103 - x**102 - x**99 + x**97 - x**96 + x**95 - x**93 + x**92 + x**91 - x**90 - x**89 + x**87 - x**86 + x**85 - x**83 + x**82 - x**81 - x

```

```

5 + 14*x**154 - 6*x**153 - 6*x**152 + 19*x**151 - 17*x**150 + 19*x**149 + 47*x**148 - 8*x**147 + 18*x**146 + 54*x**145 - 24*x**144 - 22*x**143 - 40*x**142 - 11*x**141 + 34*x**140 + 5*x**139 - 38*x**138 + 14*x**137 - 28*x**136 + 48*x**135 - 21*x**134 - 8*x**133 + 3*x**132 + 42*x**131 + 56*x**130 + 17*x**129 + 10*x**128 + 15*x**127 - 47*x**126 + 52*x**125 + 36*x**124 + 2*x**123 - 36*x**122 - 42*x**121 - 22*x**120 - 61*x**119 - 50*x**118 + 28*x**117 - 18*x**116 - 7*x**115 + 11*x**114 + 34*x**113 - 50*x**112 + 9*x**111 - 8*x**110 - 31*x**109 - 26*x**108 - 11*x**107 - 2*x**106 + 42*x**105 - 15*x**104 - 32*x**103 + 55*x**102 - 55*x**101 - 53*x**100 - 15*x**99 + 19*x**98 - 29*x**97 + 40*x**96 - 35*x**95 + 3*x**94 - 55*x**93 - 50*x**92 + 3*x**91 + 62*x**90 + 15*x**89 + 43*x**88 + 38*x**87 - 55*x**86 - 33*x**85 - 43*x**84 - 50*x**83 - 41*x**82 + 14*x**81 - 46*x**80 + 30*x**79 + 51*x**78 - 6*x**77 + 53*x**76 + 10*x**75 - 63*x**74 - 42*x**73 - 21*x**72 - 47*x**71 - 63*x**70 - 48*x**69 + 24*x**68 - 44*x**67 + 64*x**66 - 52*x**65 + 7*x**64 + 48*x**63 - 36*x**62 - 6*x**61 - 62*x**60 - 21*x**59 - 36*x**58 + 34*x**57 + 4*x**56 - 47*x**55 - 45*x**54 + 7*x**53 + 46*x**52 + 20*x**51 + 28*x**50 - 11*x**48 - 13*x**47 + 57*x**46 - 19*x**45 + 21*x**44 - 37*x**43 - 33*x**42 + 25*x**41 - 54*x**40 - 43*x**39 + 5*x**38 + 12*x**37 - 16*x**36 - 49*x**35 + 55*x**34 - 24*x**33 - 38*x**32 - 50*x**31 - 34*x**30 + x**29 - 2*x**28 - 36*x**27 - 36*x**26 + 60*x**25 - 17*x**24 + 28*x**23 - 48*x**22 - 36*x**21 + 19*x**20 + 30*x**19 - 2*x**18 + 49*x**17 - 51*x**16 - 43*x**15 + 63*x**14 - 3*x**13 - 53*x**12 + 35*x**11 + 50*x**10 + 54*x**9 - 52*x**8 - 44*x**7 + 52*x**6 + 26*x**5 + 32*x**4 - 43*x**3 - 23*x**2 - 50*x + 22, x, domain='ZZ')
Private key saved to myKey.priv file
Private key saved to myKey.pub file
$ ls
LICENSE      myKey.priv.npz  ntru          padding
README.md   myKey.pub.npz  ntru.py

```

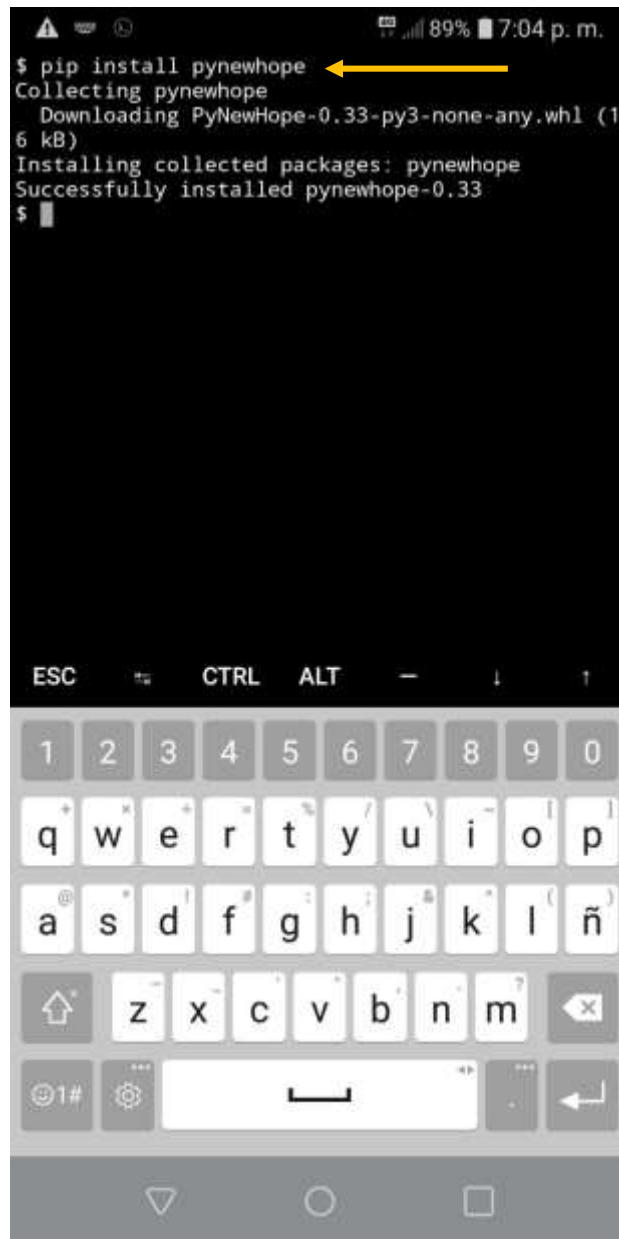
More information from NTRU: <https://ntru.org/>

Implementation of NEWHOPE algorithm.

NewHope is a key exchange protocol based on the Ring-Learning-with-errors (Ring-LWE) problem, which was sent to the NIST post-quantum crypto project.

Install NEWHOPE by executing the following command in the Termux terminal:

\$ pip install pynewhope



```
$ pip install pynewhope
Collecting pynewhope
  Downloading PyNewHope-0.33-py3-none-any.whl (16 kB)
Installing collected packages: pynewhope
Successfully installed pynewhope-0.33
$
```

More information: <https://newhopecrypto.org/>

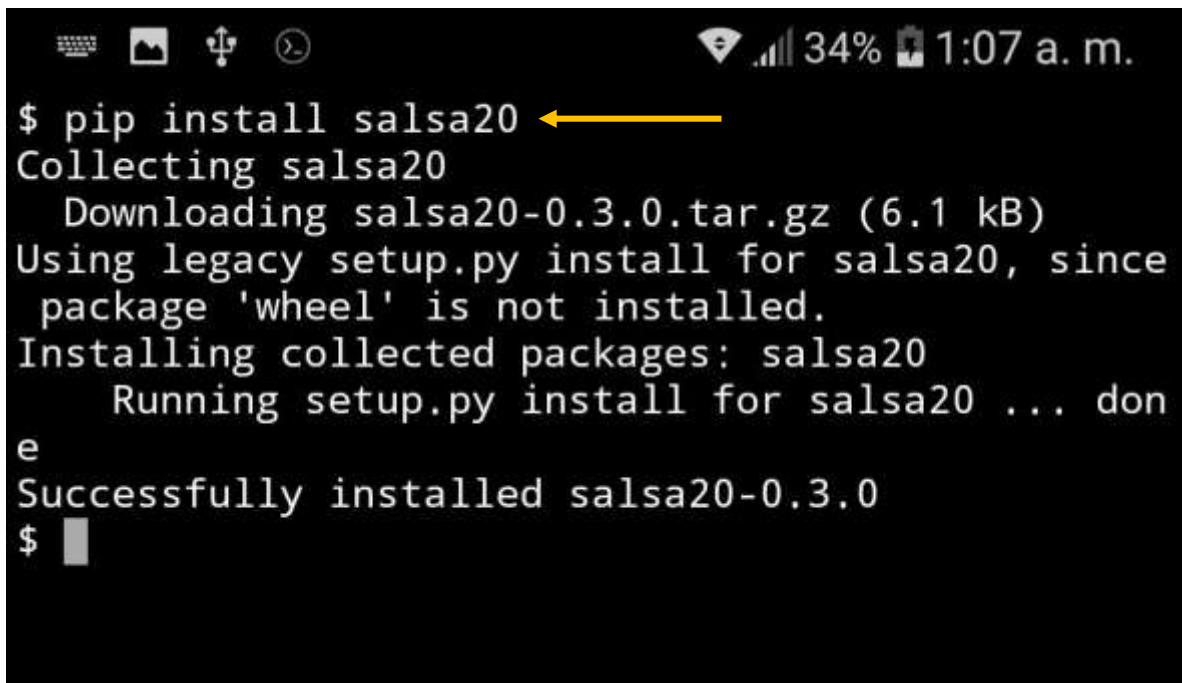
## Implementation of **SALSA20** algorithm

Salsa20 and the closely related ChaCha are flow cyphers developed by Daniel J. Bernstein. Salsa20, the original cipher, was designed in 2005, and then sent to eSTREAM by Bernstein. ChaCha is a modification of Salsa20 published in 2008. It uses a new round function that increases diffusion and increases performance on some architectures.

Both encryptions are based on a pseudo-random function based on add-rotate-XOR (ARX) operations: 32-bit addition, bit addition (XOR) and rotation operations. The main function assigns a 256-bit key, a 64-bit nonce and a 64-bit counter to a 512-bit block of the key stream (there is also a Salsa version with a 128-bit key). This gives Salsa20 and ChaCha the unusual advantage that the user can efficiently search for any position in the key stream in constant time.

Install Salsa20 by executing the following command:

**\$ apt install salsa20**

A screenshot of a mobile terminal window. The status bar at the top shows icons for keyboard, camera, USB, and a battery level of 34% at 1:07 a.m. The terminal text shows a command to install salsa20 using pip, followed by the collection and installation process. A yellow arrow points to the command line.

```
$ pip install salsa20
Collecting salsa20
  Downloading salsa20-0.3.0.tar.gz (6.1 kB)
  Using legacy setup.py install for salsa20, since package 'wheel' is not installed.
Installing collected packages: salsa20
  Running setup.py install for salsa20 ... done
Successfully installed salsa20-0.3.0
$
```

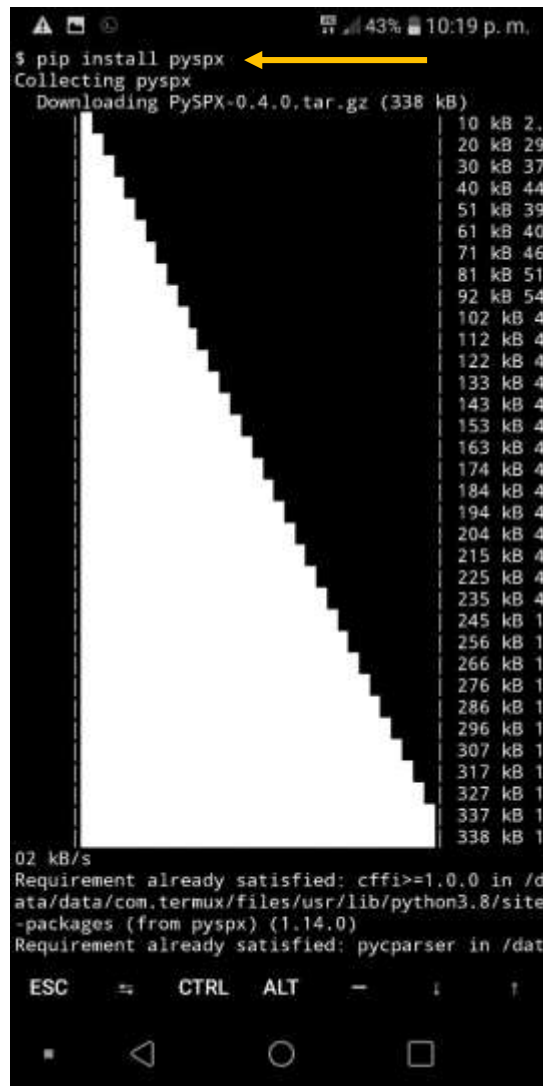
More information: <https://cr.yp.to/snuffle/salsafamily-20071225.pdf>

## Implementation of SPHINCS+ algorithm

SPHINCS + is a stateless hash-based signature scheme that was submitted to the NIST post-quantum crypto project. The design advances the SPHINCS signature scheme, which was presented at EUROCRYPT 2015. It incorporates multiple improvements, specifically aimed at reducing the size of the signature. For a quick overview of the changes from SPHINCS to SPHINCS+.

Install SPHINCS+ by executing the following command:

\$ pip install pyspx



```
$ pip install pyspx
Collecting pyspx
  Downloading PySPX-0.4.0.tar.gz (338 kB)
    10 kB 2.
    20 kB 29
    30 kB 37
    40 kB 44
    51 kB 39
    61 kB 40
    71 kB 46
    81 kB 51
    92 kB 54
    102 kB 4
    112 kB 4
    122 kB 4
    133 kB 4
    143 kB 4
    153 kB 4
    163 kB 4
    174 kB 4
    184 kB 4
    194 kB 4
    204 kB 4
    215 kB 4
    225 kB 4
    235 kB 4
    245 kB 1
    256 kB 1
    266 kB 1
    276 kB 1
    286 kB 1
    296 kB 1
    307 kB 1
    317 kB 1
    327 kB 1
    337 kB 1
    338 kB 1
02 kB/s
Requirement already satisfied: cffi>=1.0.0 in /data/data/com.termux/files/usr/lib/python3.8/site-packages (from pyspx) (1.14.0)
Requirement already satisfied: pycparser in /data/data/com.termux/files/usr/lib/python3.8/site-packages (from cffi>=1.0.0->pyspx) (2.21)
```

More information: <https://sphincs.org/>



Implementation of **RAINBOW** algorithm

A quantum resistive asymmetric key generation tool based on the RAINBOW scheme for digital signatures.

Digital signatures are a common way to authenticate a sender by verifying information attached to the document. In our case, the attached information is a sequence of numbers generated using the document to be signed.

As mentioned above, we use the Generalized Unbalanced Oil and Vinegar scheme or the Rainbow scheme to achieve the same.

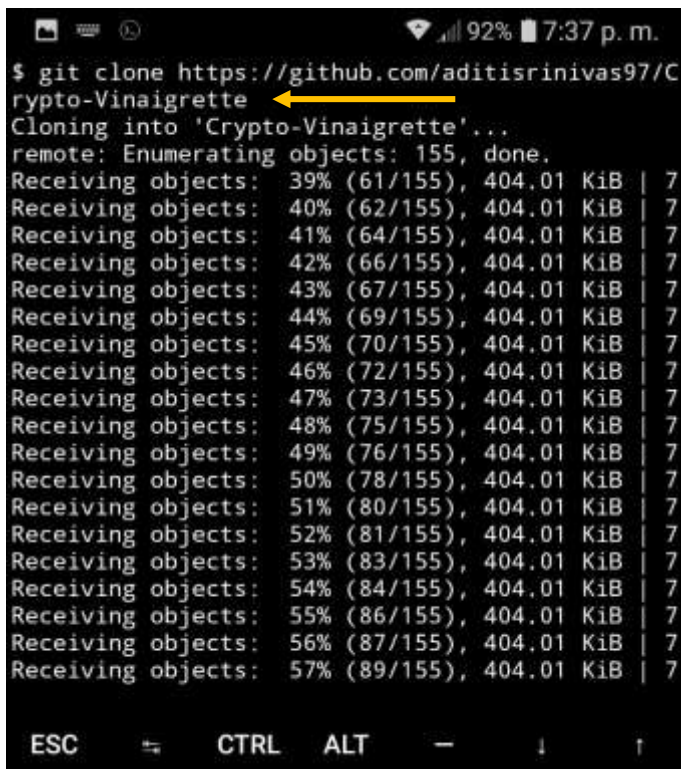
Before installing, we need to have the "git" package installed with: `$ apt install git`

Install RAINBOW by executing the following commands:

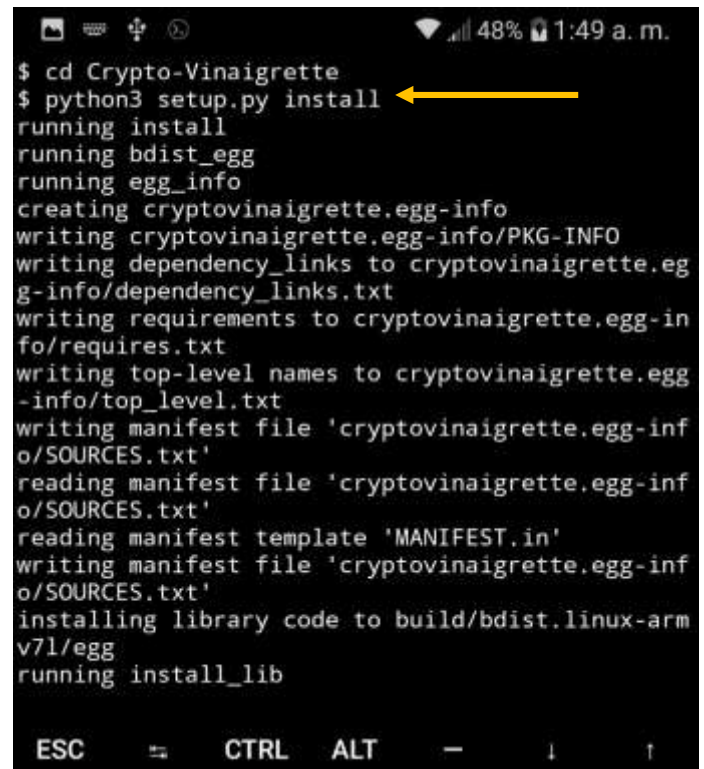
```
$ git clone https://github.com/aditisrinivas97/Crypto-Vinaigrette
```

```
$ cd Crypto-Vinaigrette
```

```
$ python3 setup.py install
```

A terminal window with a black background and white text. The status bar at the top shows battery at 92% and time at 7:37 p.m. The command `$ git clone https://github.com/aditisrinivas97/Crypto-Vinaigrette` has been executed. The output shows the cloning progress, with a yellow arrow pointing to the repository name 'Crypto-Vinaigrette'. The progress bar indicates 57% completion (89/155 objects).

```
$ git clone https://github.com/aditisrinivas97/Crypto-Vinaigrette
Cloning into 'Crypto-Vinaigrette'...
remote: Enumerating objects: 155, done.
Receiving objects: 39% (61/155), 404.01 KiB | 7
Receiving objects: 40% (62/155), 404.01 KiB | 7
Receiving objects: 41% (64/155), 404.01 KiB | 7
Receiving objects: 42% (66/155), 404.01 KiB | 7
Receiving objects: 43% (67/155), 404.01 KiB | 7
Receiving objects: 44% (69/155), 404.01 KiB | 7
Receiving objects: 45% (70/155), 404.01 KiB | 7
Receiving objects: 46% (72/155), 404.01 KiB | 7
Receiving objects: 47% (73/155), 404.01 KiB | 7
Receiving objects: 48% (75/155), 404.01 KiB | 7
Receiving objects: 49% (76/155), 404.01 KiB | 7
Receiving objects: 50% (78/155), 404.01 KiB | 7
Receiving objects: 51% (80/155), 404.01 KiB | 7
Receiving objects: 52% (81/155), 404.01 KiB | 7
Receiving objects: 53% (83/155), 404.01 KiB | 7
Receiving objects: 54% (84/155), 404.01 KiB | 7
Receiving objects: 55% (86/155), 404.01 KiB | 7
Receiving objects: 56% (87/155), 404.01 KiB | 7
Receiving objects: 57% (89/155), 404.01 KiB | 7
```

A terminal window with a black background and white text. The status bar at the top shows battery at 48% and time at 1:49 a.m. The command `$ python3 setup.py install` has been executed. The output shows the installation progress, with a yellow arrow pointing to the command. The progress bar indicates 100% completion.

```
$ cd Crypto-Vinaigrette
$ python3 setup.py install
running install
running bdist_egg
running egg_info
creating cryptovinaigrette.egg-info
writing cryptovinaigrette.egg-info/PKG-INFO
writing dependency_links to cryptovinaigrette.egg-info/dependency_links.txt
writing requirements to cryptovinaigrette.egg-info/requirements.txt
writing top-level names to cryptovinaigrette.egg-info/top_level.txt
writing manifest file 'cryptovinaigrette.egg-info/SOURCES.txt'
reading manifest file 'cryptovinaigrette.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
writing manifest file 'cryptovinaigrette.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-armv7l/egg
running install_lib
```

More information: <https://bit.ly/2vwckRw>

Implementation of **CODECRYPT** algorithm.

CODECRYPT. - It is a tool that encompasses different PQC (Post-Quantum Cryptography) algorithms to perform the main processes of encryption and decryption of text and files.

Classified with the following identifiers: [S]ign, [E]ncryption, [C]ipher and [H]ash.

S FMTSEQ128C-CUBE256-CUBE128  
S FMTSEQ128C-SHA256-RIPEMD128  
S FMTSEQ128H20C-CUBE256-CUBE128  
S FMTSEQ128H20C-SHA256-RIPEMD128  
S FMTSEQ192C-CUBE384-CUBE192  
S FMTSEQ192C-SHA384-TIGER192  
S FMTSEQ192H20C-CUBE384-CUBE192  
S FMTSEQ192H20C-SHA384-TIGER192  
S FMTSEQ256C-CUBE512-CUBE256  
S FMTSEQ256C-SHA512-SHA256  
S FMTSEQ256H20C-CUBE512-CUBE256  
S FMTSEQ256H20C-SHA512-SHA256  
E MCEQCMDPC128FO-CUBE256-ARCFOUR  
E MCEQCMDPC128FO-CUBE256-CHACHA20  
E MCEQCMDPC128FO-CUBE256-XSYND  
E MCEQCMDPC128FO-SHA256-ARCFOUR  
E MCEQCMDPC128FO-SHA256-CHACHA20  
E MCEQCMDPC128FO-SHA256-XSYND  
E MCEQCMDPC256FO-CUBE512-ARCFOUR  
E MCEQCMDPC256FO-CUBE512-CHACHA20  
E MCEQCMDPC256FO-CUBE512-XSYND  
E MCEQCMDPC256FO-SHA512-ARCFOUR  
E MCEQCMDPC256FO-SHA512-CHACHA20  
E MCEQCMDPC256FO-SHA512-XSYND  
C ARCFOUR  
C CHACHA20  
C XSYND  
H CUBE512  
H RIPEMD128  
H SHA256  
H SHA512  
H TIGER192

Now we will carry out the installation of the **CODECRYPT** suite of tools with the execution of the following command in the termux terminal, an important point is that with this execution *two dependencies cryptopp and fftw* will be installed:

```
$ apt install codecrypt
```



```
$ apt install codecrypt
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cryptopp fftw
The following NEW packages will be installed:
  codecrypt cryptopp fftw
0 upgraded, 3 newly installed, 0 to remove and 0
not upgraded.
Need to get 2065 kB of archives.
After this operation, 9208 kB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
```



```
Building dependency tree
Reading state information... Done
The following additional packages will be instal
led:
  cryptopp fftw
The following NEW packages will be installed:
  codecrypt cryptopp fftw
0 upgraded, 3 newly installed, 0 to remove and 0
not upgraded.
Need to get 2065 kB of archives.
After this operation, 9208 kB of additional disk
space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm cryptopp arm 8.2.0-5 [118
8 kB]
Get:2 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm fftw arm 3.3.8-2 [756 kB]
Get:3 https://dl.bintray.com/termux/termux-packa
ges-24 stable/main arm codecrypt arm 1.8-8 [121
kB]
Fetched 2065 kB in 2s (804 kB/s)
Selecting previously unselected package cryptopp
.
(Reading database ... 3449 files and directories
currently installed.)
Preparing to unpack .../cryptopp_8.2.0-5_arm.deb
...
Unpacking cryptopp (8.2.0-5) ...
Selecting previously unselected package fftw.
Preparing to unpack .../archives/fftw_3.3.8-2_ar
m.deb ...
Unpacking fftw (3.3.8-2) ...
Selecting previously unselected package codecryp
t.
Preparing to unpack .../codecrypt_1.8-8_arm.deb
...
Unpacking codecrypt (1.8-8) ...
Setting up fftw (3.3.8-2) ...
Setting up cryptopp (8.2.0-5) ...
Setting up codecrypt (1.8-8) ...
$
```



## 8. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor is a software development environment created by Google Labs to build applications for the Android operating system. The user can, visually and from a set of basic tools, link a series of blocks to create the application. The system is free and can be easily downloaded from the web. Applications created with App Inventor are very easy to create because no knowledge of any programming language is required.

All the current environments that use Blockly's technology such as AppyBuilder and Thunkable among others have their free version, their way of use can be through the internet in their different sites or it can also be installed at home.

The blocks that make up the Mini BloclyChain architecture have been tested in App inventor and AppyBuilder but because of their code optimization they should work on the other platforms.

Online versions:

App Inventor.

<https://appinventor.mit.edu/>

AppyBuilder.

<http://appybuilder.com/>

Thunkable.

<https://thunkable.com/>

Version to be installed on your computer (PC):

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

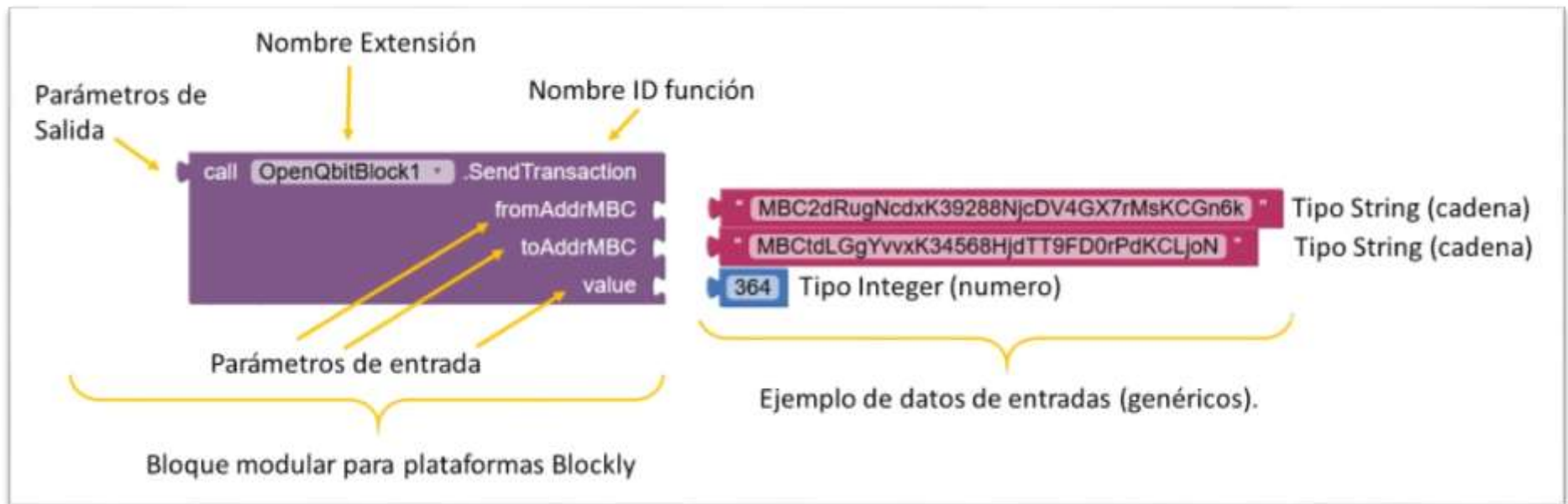
Environment for developers of Blockly blocks.

<https://editor.appybuilder.com/login.php>

## 9. Definition and use of blocks in Mini PQC

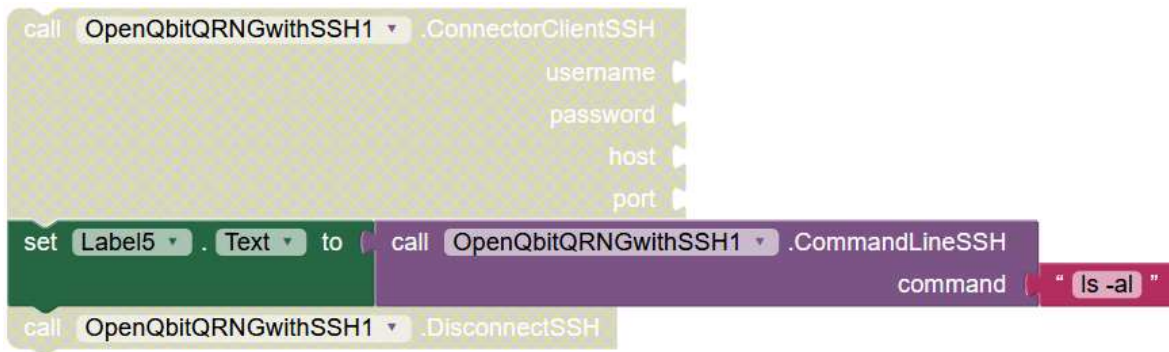
We will start by explaining the distribution of the data that all the blocks will have, their syntax of use and configuration.

In the following example we can see a modular block and its input and output parameters, as well as the types of input data, these data can be of type String (string of characters) or Integer (integer or decimal). We show how it is used and configure it for its proper functioning.



Each module block will have its description and will be named in case it has any mandatory or optional dependency(s) of other blocks used as input parameters, the integration process will be announced. Let's start with the blocks of the **OpenQbitPQCwithSSH** extension.

Block to execute command in Termux terminal - (**CommandLineSSH**)



Input parameters: **command** <String>

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: Execute the command entered in the Termux terminal.

Description: An entered command is executed and the block is needed first to connect to the SSH server (**ConnectClientSSH**) and then to use the block (**DisconnectSSH**).

Block to connect to a remote or local SSH server - (**ConnectorClientSSH**).

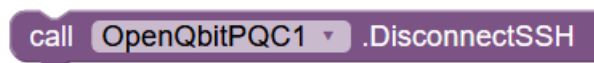


Input parameters: **username** <string>, **password** <string>, **host** <string>, **port** <integer>

Output parameters: If the connection with the ssh server of the Termux terminal is successful, it gives us a message; **"Connect SSH"**, if it is not successful, it gives us a **NULL** message.

Description: Communication block to connect the chosen SSH server to the Termux terminal, via SSH (Secure Shell) communication protocol.

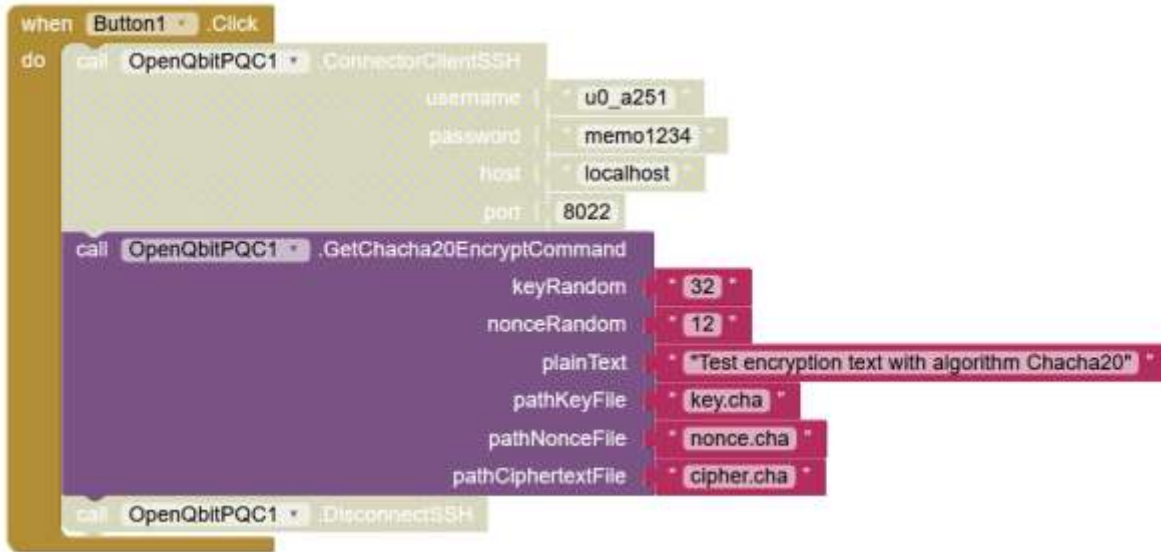
Block to close the SSH session (**DisconnectSSH**).



Input and output parameters: Not applicable (none)

Description: Block to close SSH session.

Block for encrypting character strings with Post-Quantum Cryptography PQC algorithm Chacha20Poly1302 - (**GetChacha20EncryptCommand**)



Parámetros de entrada: **keyRandom** <String>, **nonceRandom** <String>, **plaintext** <String>, **pathKeyFile** <String>, **pathNonceFile** <String>, **pathCiphertextFile** <String>

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

The parameters keyRandom and nonceRandom are integers for the generation of pseudo-random numbers, if you want random numbers you can use the extension (**OpenQbitQRNGwithSSH**), in our example we use pseudo-allotories.

Output parameters: Execute the event (**OutputDataChacha20**). It delivers a base64 encrypted string.



Ejemplo: "Test encryption text with algorithm Chacha20"

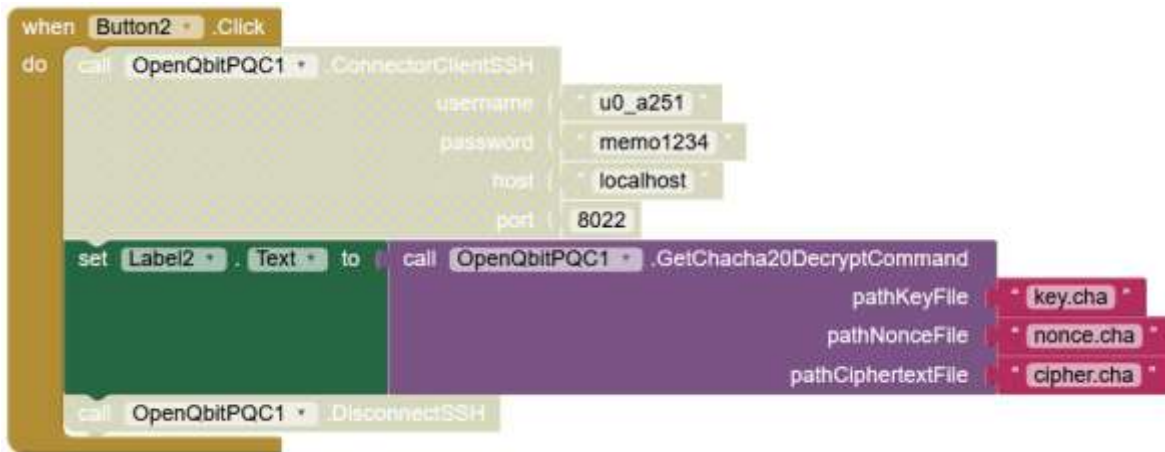
Output:

OCTxa3YlrpOUKRUn3tpcsjU7w0ZckO/4FMx0/Ybsz3hISOXNBY/G0B8Jf6FAVrwz135q+h6A4f/m  
Ooi

Three files are generated for use in the decryption process in the specified path. The names can be user selected, remembering that each time the block is executed if it is not renamed it will be rewritten by the new encrypted message.

Description: Block that encrypts a string of characters with the Chacha20Poly1305 algorithm applying PCQ (Post-Quantum Cryptography).

Block for deciphering character strings with Post-Quantum Cryptography PQC algorithm Chacha20Poly1302 - (**GetChacha20DecryptCommand**)



Input parameters: **keyRandom** <String>, **nonceRandom** <String>, **pathCiphertextFile** <String>

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: Delivers the decrypted string containing the file in the specified **pathCiphertextFile** this file was generated in the encryption block (**GetChacha20EncryptCommand**).

Example:

OCTxa3YlrpOUKRUn3tpcsjU7w0ZckO/4FMx0/Ybsz3hISOXNBY/G0B8Jf6FAVrwz135q+h6A4f/m  
Ooi

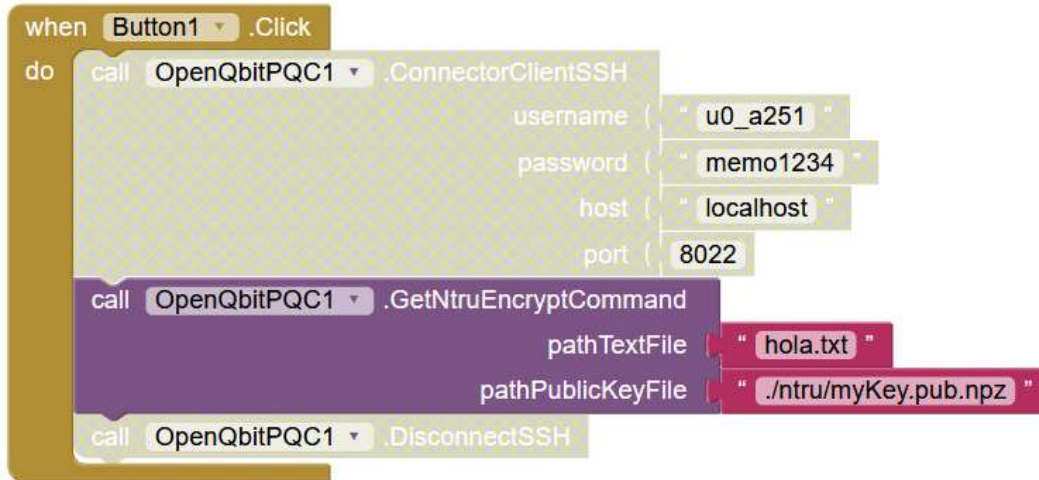
Output: "Test encryption text with algorithm Chacha20"

### IMPORTANT NOTE:

The previous blocks (**GetChacha20EncryptCommand**) and (**GetChacha20DecryptCommand**) before using them you must install in Termux's terminal the programs **chacha20encrypt.py** and **chacha20decrypt.py** and place them in Termux's default user home which is: **/data/data/com.termux/files/home**, this user is the one you are connecting with via SSH (Secure Shell)

See Annex "Python Programs".

Block for encrypting a plain text file with Post-Quantum Cryptography PQC NTRU algorithm - (GetNtruEncryptCommand)



Input parameters: **pathTextFile**< String>, **pathPublicKeyFile**< String>

Mandatory dependency: Block (ConnectorClientSSH), Block (DisconnectSSH).

Output parameters: Delivers a file **encrypted** with the start word: **encrypted**

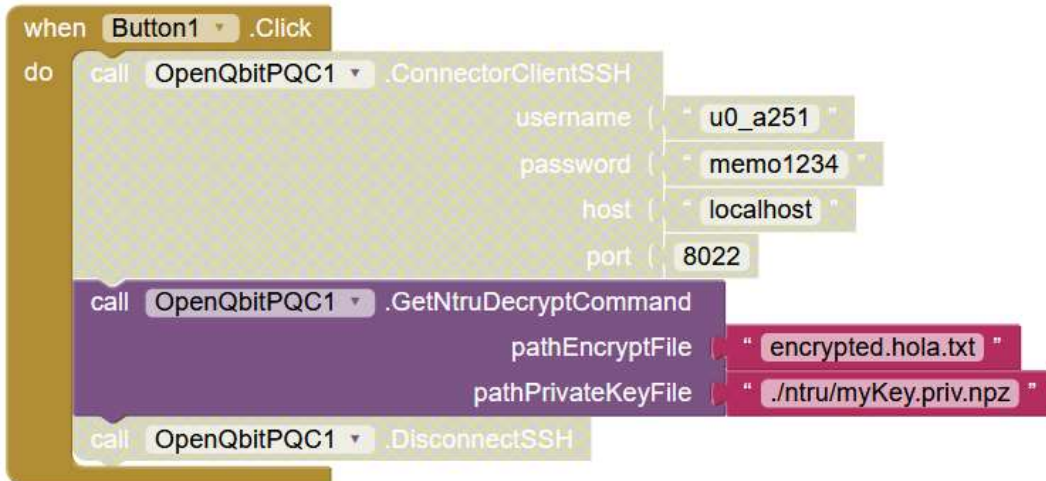
In our case, the file hello.txt was introduced, the result would be a file with the name:

**encrypted.hello.txt**

Description: Encrypt small text files. **txt** these must be **100 characters maximum**.

Block to **decrypt** a text file with Post-Quantum Cryptography PQC NTRU algorithm - (GetNtruDecryptCommand)

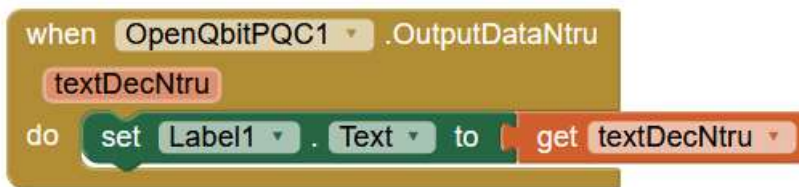




Input parameters: **pathEncryptFile<String>**, **pathPrivateKeyFile<String>**

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: The event (**OutputDataNtru**) is executed with the output parameter **TexDectNtru** which is the original decrypted text contained in the text file.

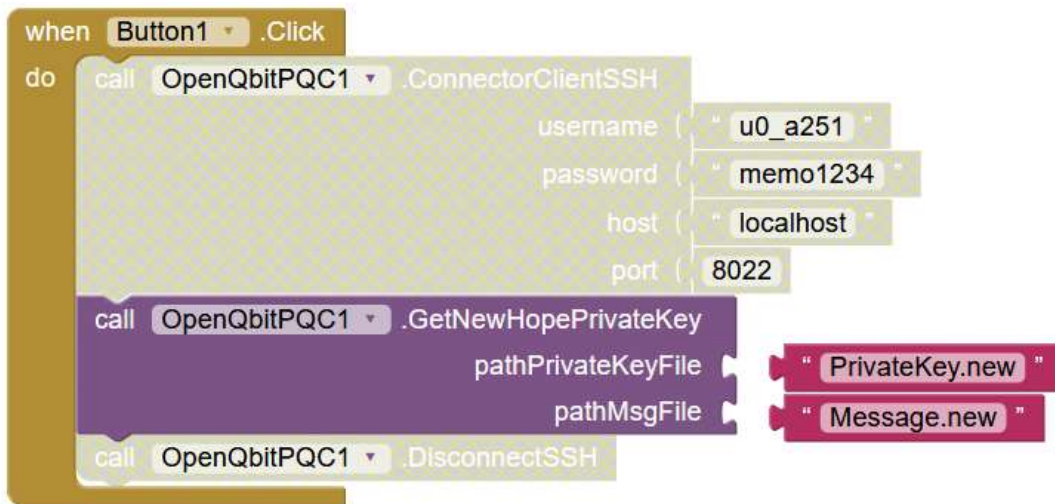


In our case, the file hello.txt was introduced, the result would be a file with the name:

**encrypted.hello.txt**

Description: Encrypt small text files which should be no **longer than 100 characters**.

Block to generate private key with NEWHOPE algorithm (**GetNewHopePrivateKey**)



Input parameters: **pathNewHopePrivateKey<String>**, **pathMsgFile<String>**

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: The event (**OutputDataNewHopePrivateKeyandMsg**) is executed with the output parameter **cipherNewText** which is the content of the file given in the input parameter **pathNewHopePrivateKey** concatenated with the content of the input parameter **pathMsgFile**.



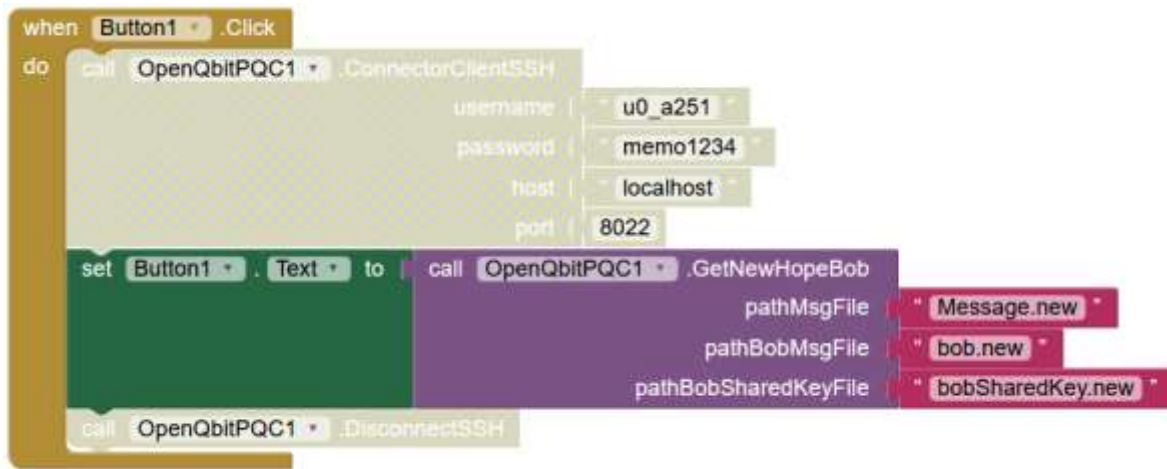
The output has the following format:

"----PRIVATE KEY----", PrivateKey Value, " ----MESSAGE----", Message Value

Description: Generates a private key and a message and saves them in two separate files according to the "path" given in the input parameters, which will be used later to use it between two users to verify the keys to be shared.



Block to generate generic user key "Bob" (**GetNewHopeBob**)



Input parameters: **pathMsgFile<String>**, **pathBobMsgFile<String>**, **pathBobSharedKeyFile<String>**

Required dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**), Block (**GetNewHopePrivateKey**).

Output parameters: The files given in the parameters **pathBobMsgFile** and **pathBobSharedKeyFile** are created and an output of the file of the name given in the input parameter **pathBobSharedKeyFile** is delivered.

The output has the content of the file of the parameter **pathBobSharedKeyFile** with the following format

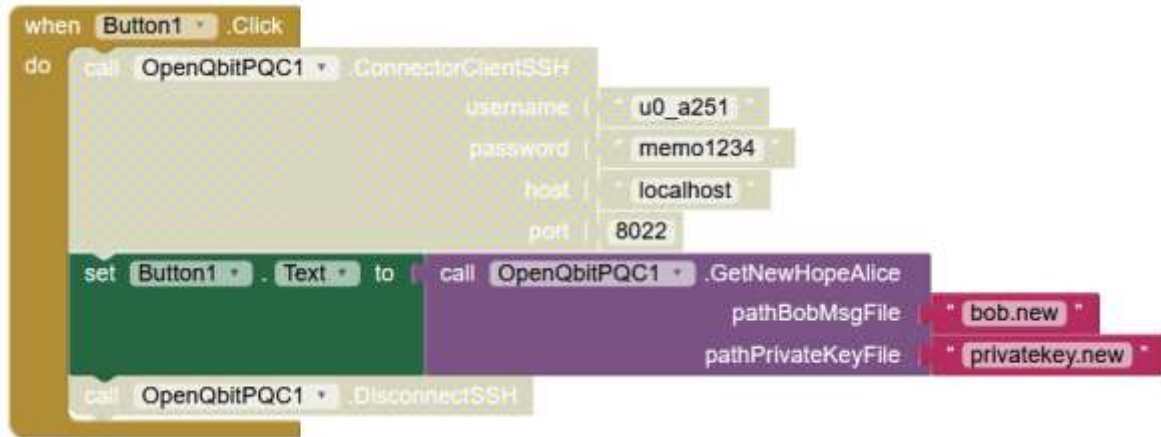
"----- SHARED KEY BOB----- ", content of the pathBobSharedKeyFile

Example:

```
----- SHARED KEY BOB----- [92, 122, 75, 33, 239, 164, 84, 241, 245, 204,
106, 197, 142, 230, 28, 189, 54, 112, 190, 124, 176, 66, 129, 69, 108,
66, 110, 42, 115, 70, 17, 107]
```

Description: Block to generate key to be shared among users.

Block to generate generic user key "Alice" (**GetNewHopeAlice**)



Input parameters: **pathBobMsgFile**< String>, **pathPrivateKeyFile**< String>

Required dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**), Block (**GetNewHopeBob**).

Output parameters: The sharing key of the second user is created. In this case we use a generic name "Alice".

The output has the content of the key to be shared with the following format:

"-----SHARED KEY ALICE-----", **aliceSharedKey**

Example:

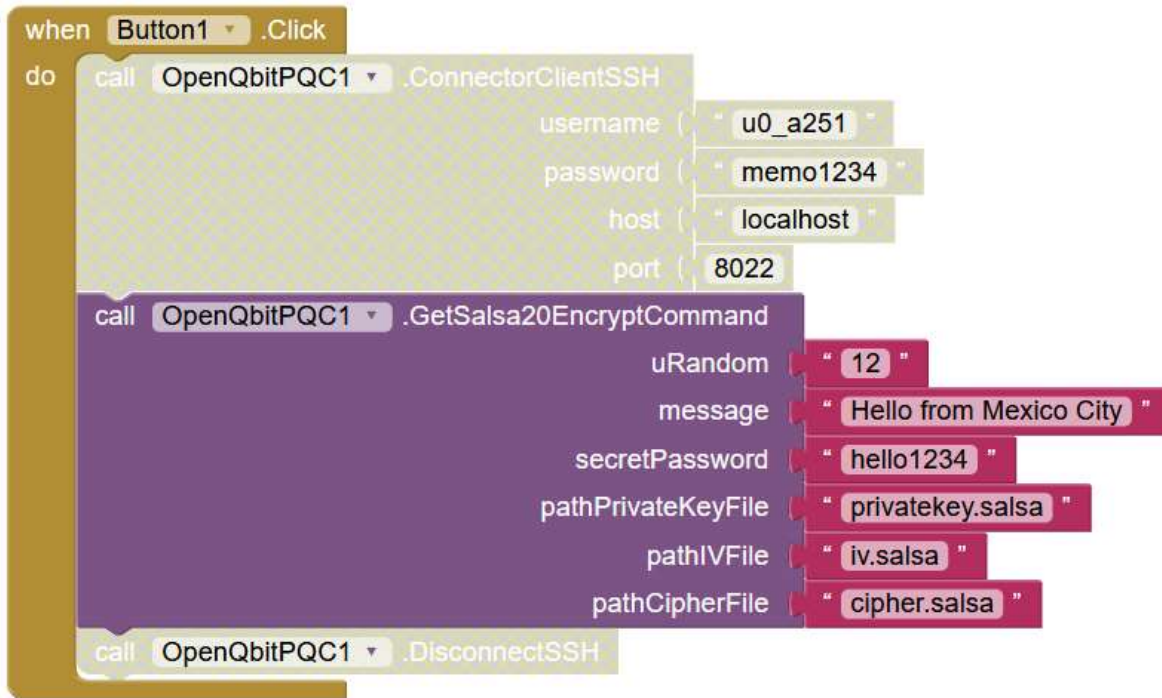
```
----- SHARED KEY ALICE----- [92, 122, 75, 33, 239, 164, 84, 241, 245,
204, 106, 197, 142, 230, 28, 189, 54, 112, 190, 124, 176, 66, 129, 69,
108, 66, 110, 42, 115, 70, 17, 107]
```

Description: Block to generate key to be shared among users.

**CONFIRMATION OF NEWHOPE:** To confirm that the key exchange is correct, the results of the (**GetNewHopeBob**) and (**GetNewHopeAlice**) bots must be the same.

The NEWHOPE algorithm is just an **exchange of keys** (lists of numbers) that helps to validate information sent and can for example be used as a password for an encryption method such as AES.

Block to encrypt message with Salsa20 algorithm (**GetSalsa20EncryptCommand**)



Input parameters: **uRandom<String>**, **message<String>**, **secretPassword<String>**, **pathPrivateKeyFile<String>**, **pathIVFile<String>**, **pathCipherFile<String>**

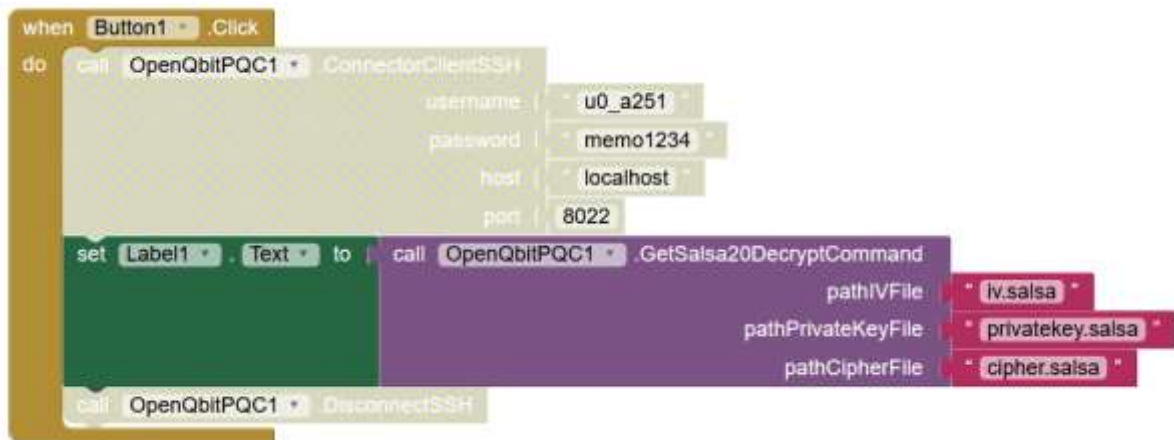
Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: The event (**OutputDataSalsa20**) is executed with the output parameter **cipherTextS** which is the encrypted message.



Description: Block to encrypt a message with Salsa20 algorithm.

Block to message decryption tool with Salsa20 algorithm (**GetSalsa20DecryptCommand**)



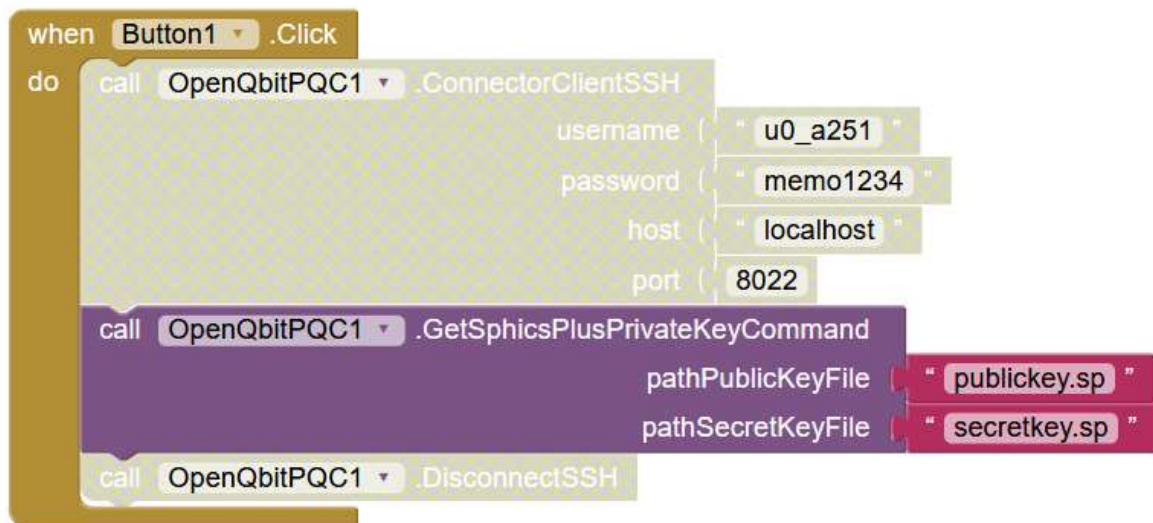
Input parameters: **pathIVFile<String>**, **pathPrivateKeyFile<String>** , **pathCipherFile<String>**

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: Delivery in original decrypted output

Description: Block to decode a message with Salsa20 algorithm.

Block to generate private key with Sphics+ algorithm (**GetSphicsPlusPrivateKey**)



Input parameters: **pathPublicKeyFile< String>**, **pathSecretKeyFile< String>**

Mandatory dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**).

Output parameters: The event (**OutputDataSphicsPlus**) is executed with the output parameter cipherTextSP that delivers the public key and secret key concatenated.

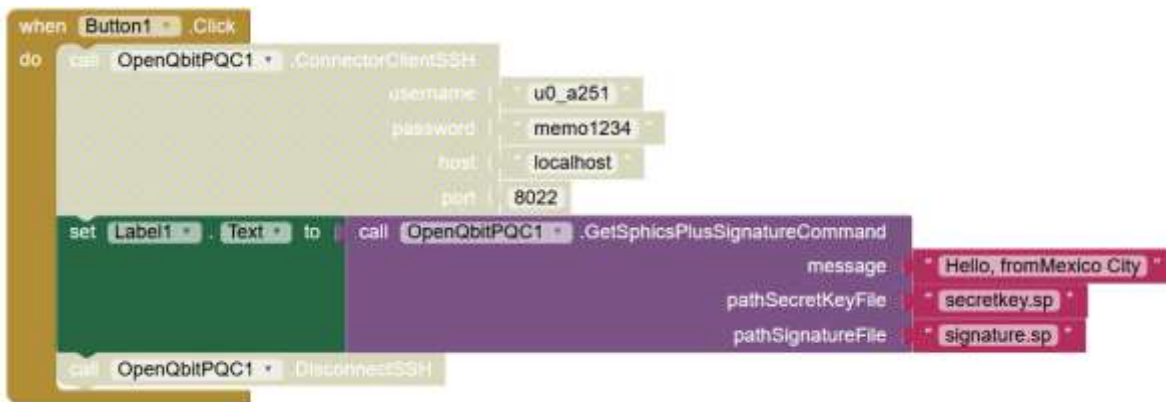


The output format is:

```
"----PUBLIC KEY----",public_key,"----SECRET KEY----", secret_key
```

Description: Generates a private key and a secret key and saves them in two separate files according to the path given in the input parameters.

Block to sign a message with the SpHics+ algorithm (**GetSpHicsPlusSignature**)



Input parameters: **message<String>**, **pathSecretKeyFile<String>**, **pathSignatureFile<String>**

Required dependency: Block (**ConnectorClientSSH**), Block (**DisconnectSSH**), Block (**GetSpHicsPlusPrivateKey**).

Output parameters: it creates the file with the signature of the message that was given in the message parameter.

Description: Creates a signature that identifies the message provided.

## 10. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor is a software development environment created by Google Labs to build applications for the Android operating system. The user can, visually and from a set of basic tools, link a series of blocks to create the application. The system is free and can be easily downloaded from the web. Applications created with App Inventor are very easy to create because no knowledge of any programming language is required.

All the current environments that use Blockly's technology such as AppyBuilder and Thunkable among others have their free version, their way of use can be through the internet in their different sites or it can also be installed at home.

The blocks that make up the Mini BloclChain architecture have been tested in App inventor and AppyBuilder but because of their code optimization they should work on the other platforms.

Online versions:

App Inventor.

<https://appinventor.mit.edu/>

AppyBuilder.

<http://appybuilder.com/>

Thunkable.

<https://thunkable.com/>

Version to be installed on your computer (PC):

<https://sites.google.com/site/aprendeappinventor/instala-app-inventor>

Environment for developers of Blockly blocks.

<https://editor.appybuilder.com/login.php>

## 11. Annex "Python programs".

*The names and locations should not be changed since the command block refers to this name and location.*

Programs for Chacha20Poly1305 algorithm that should be installed in the Termux user home default `/data/data/com.termux/files/home` of the Termux terminal user.

Name: `chacha20encrypt.py`

```

imports
import sys
import binascii
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode
from chacha20poly1305 import ChaCha20Poly1305

k=int(sys.argv[1])
n=int(sys.argv[2])
key = os.urandom(k)
st=sys.argv[3]
abin = bytearray(st, "utf8")
nonce = os.urandom(n)
cip = ChaCha20Poly1305(key)

ciphertext = cip.encrypt(nonce, abin)
keyBase64=standard_b64encode(key).decode("utf-8")
with open(sys.argv[4], 'w') as f:
    f.write(keyBase64)

nonceBase64=standard_b64encode(nonce).decode("utf-8")
with open(sys.argv[5], 'w') as f:
    f.write(nonceBase64)

cipherBase64=standard_b64encode(ciphertext).decode("utf-8")
with open(sys.argv[6], 'w') as f:
    f.write(cipherBase64)

print(cipherBase64)

```

Name: `chacha20decrypt.py`

```
imports
import sys
import binascii
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode
from chacha20poly1305 import ChaCha20Poly1305

inputCipher = open(sys.argv[3], 'r')
bufferCipher = inputCipher.read()
inputCipher.close

inputNonce = open(sys.argv[2], 'r')
bufferNonce = inputNonce.read()
inputNonce.close

inputKey = open(sys.argv[1], 'r')
bufferKey = inputKey.read()
inputKey.close

cipher=standard_b64decode(bufferCipher)
nonceFile=standard_b64decode(bufferNonce)
keyFile=standard_b64decode(bufferKey)

cip = ChaCha20Poly1305(keyFile)
plaintext = cip.decrypt(nonceFile, cipher).decode("utf-8")
print(plaintext)
```



*The names and locations should not be changed since the command block refers to this name and location.*

Programs for NEWHOPE algorithm to be installed in the Termux default user home `/data/data/com.termux/files/home` of the Termux terminal user.

Name: **newhope-privatekey.py**

```
from pynewhope import newhope
import pickle
import sys

# Step 1: Alice generates random keys and her public msg to Bob
PrivKey, Msg = newhope.keygen()

with open(sys.argv[1], 'wb') as ft:
    pickle.dump(PrivKey, ft)

with open(sys.argv[2], 'wb') as ft:
    pickle.dump(Msg, ft)

print("----- PRIVATE KEY----- ", PrivKey, "----- MESSAGE----- ", Msg)
```

Name: **newhope-bob.py**

```
from pynewhope import newhope
import pickle
import sys

with open(sys.argv[1], 'rb') as ft:
    Msg = pickle.load(ft)

bobSharedKey, bobMsg = newhope.sharedB(Msg)

with open(sys.argv[2], 'wb') as ft:
    pickle.dump(bobMsg, ft)

with open(sys.argv[3], 'wb') as ft:
    pickle.dump(bobSharedKey, ft)

print("----- SHARED KEY BOB----- ", bobSharedKey)
```

Name: **newhope-alice.py**

```
from pynewhope import newhope
import pickle
import sys

with open(sys.argv[1], 'rb') as fp:
    bobMsg = pickle.load(fp)

with open(sys.argv[2], 'rb') as fp:
    PrivKey = pickle.load(fp)

AliceSharedKey = newhope.sharedA(bobMsg, PrivKey)

print("----- SHARED KEY ALICE----- ", aliceSharedKey)
```

Create in App inventor the following comparative to confirm that the key exchange between Bob and Alice (example users name) is correct.

```
if aliceSharedKey == bobSharedKey:
    print("\nSuccessful key exchange! Keys match.")
else:
    print("Error! Keys do not match.")
```

*The names and locations should not be changed since the command block refers to this name and location.*

Programs for **SALSA20** algorithm that should be installed in the Termux default user home **/data/data/com.termux/files/home** of the Termux terminal user.

Name: **salsa20encrypt.py**

```
from salsa20 import XSalsa20_xor
from os import urandom
import binascii
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys

k=int(sys.argv[1])
msg=sys.argv[2]
secretKEY=sys.argv[3]
IV = os. urandom(k)
KEY = bytearray(secretKEY, "utf8")
Message= bytearray(msg, "utf8")

keyBase64=standard_b64encode(KEY).decode("utf-8")
with open(sys.argv[4], 'w') as f:
    f.write(keyBase64)

ivBase64=standard_b64encode(IV).decode("utf-8")
with open(sys.argv[5], 'w') as f:
    f.write(ivBase64)

ciphertext = XSalsa20_xor(Message, IV, KEY)

ivBase64=standard_b64encode(ciphertext).decode("utf-8")
with open(sys.argv[6], 'w') as f:
    f.write(ivBase64)

print(ciphertext)
```

Name: salsa20decrypt.py

```
from salsa20 import XSalsa20_xor

import binascii
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys

inputCipher = open(sys.argv[1], 'r')
bufferCipher = inputCipher.read()
inputCipher.close

inputIV = open(sys.argv[2], 'r')
bufferIV = inputIV.read()
inputIV.close

inputKey = open(sys.argv[3], 'r')
bufferKey = inputKey.read()
inputKey.close

ciphertext=standard_b64decode(bufferCipher)
ivFile=standard_b64decode(bufferIV)
keyFile=standard_b64decode(bufferKey)

print(XSalsa20_xor(ciphertext, ivFile, keyFile).decode())
```

*The names and locations should not be changed since the command block refers to this name and location.*

Programs for SPHINCS+ algorithm to be installed in the Termux default user home /data/data/com.termux/files/home of the Termux terminal user.

Name: sphicsplus-privatekey.py

```
import pyspx.shake256_128f
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys
seed=int(sys.argv[1])

seed = os.urandom(sphincs.crypto_sign_SEEDBYTES)
public_key, secret_key = pyspx.shake256_128f.generate_keypair(seed)
```

```
keyBase64=standard_b64encode(public_key).decode("utf-8")
with open(sys.argv[2], 'w') as f:
    f.write(keyBase64)

keyBase64=standard_b64encode(secret_key).decode("utf-8")
with open(sys.argv[3], 'w') as f:
    f.write(keyBase64)

print("----PUBLIC KEY----",public_key,"----SECRET KEY----", secret_key)
```

Name: sphicsplus-signature.py

```
import pyspx.shake256_128f
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys

message=sys.argv[1]

inputSecretkey = open(sys.argv[2], 'r')
bufferSecretkey = inputSecretkey.read()
inputSecretkey.close

secret_key=standard_b64decode(bufferSecretkey)

signature = pyspx.shake256_128f.sign(message, secret_key)

keyBase64=standard_b64encode(signature).decode("utf-8")
with open(sys.argv[3], 'w') as f:
    f.write(keyBase64)

print(signature)
```

Name: sphicsplus-verify.py

```
import pyspx.shake256_128f
import base64
import hashlib
from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys
```

```
message=sys.argv[1]

inputSign = open(sys.argv[2], 'r')
bufferSign = inputSign.read()
inputSign.close

inputKey = open(sys.argv[3], 'r')
bufferKey = inputKey.read()
inputKey.close

signature=standard_b64decode(bufferSign)
public_key=standard_b64decode(bufferKey)

pyspx.shake256_128f.verify(message, signature, public_key)
True
```

*The names and locations should not be changed since the command block refers to this name and location.*

Programs for **RAINBOW** algorithm to be installed in the Termux default user home **/data/data/com.termux/files/home** of the Termux terminal user.

Name: rainbow-privatekey.py

```
from cryptovinaigrette import cryptovinaigrette

# Initialise keygen object and generate keys
folder= sys.argv[1]
myKeyObject = cryptovinaigrette.rainbowKeygen(save=folder)
```

Name: rainbow-signature.py

```
from cryptovinaigrette import cryptovinaigrette
import base64
import hashlib
from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys

filePrivateKey=sys.argv[1]
fileSigned=sys.argv[2]

signature = cryptovinaigrette.rainbowKeygen.sign(sys.argv[1],
sys.argv[2])

keyBase64=standard_b64encode(signature).decode("utf-8")
with open(sys.argv[3], 'w') as f:
    f.write(keyBase64)

Print(signature)
```

Name: rainbow-verify.py

```
from cryptovinaigrette import cryptovinaigrette
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

imports
import sys

filePrivateKey=sys.argv[1]
fileSigned=sys.argv[2]


inputSign = open(sys.argv[3], 'r')
bufferSign = inputSign.read()
inputSign.close
signature=standard_b64decode(bufferSign)

#Case where signature is valid
check = cryptovinaigrette. rainbowKeygen. verify(sys.argv[1], signature,
sys.argv[2])

if check == True :
    print("Verified successfully!")
else :
    print("Signature does not match the file!")
```



**Encrypting** files with **CODECRYPT** this suite has a variety of Post-Quantum Computing (PQC) algorithms. We will use it directly on the command line through the SSH (Secure Shell) service, with block (**OpenQbitConnectSSH**).

Example of the use of commands:

```
ccr -g help
ccr -g sig --name "John Doe"      # your signature key
ccr -g enc --name "John Doe"      # your encryption key

ccr -K #watch the generated keys

ccr -p -a -o my_pubkeys.asc -F Doe # export your pubkeys for friends

#(now you should exchange the pubkeys with friends)

#see what people sent us, possibly check the fingerprints
ccr -inaf < friends_pubkeys.asc

#import Frank's key and rename it
ccr -ia -R friends_pubkeys.asc --name "Friendly Frank"

#send a nice message to Frank (you can also specify him by @12345 keyid)
ccr -se -r Frank < Document.doc > Message_to_frank.ccr

#receive a reply
ccr -dv -o Decrypted_verified_reply.doc <Reply_from_frank.ccr

#rename other's keys
ccr -m Frank -N "Unfriendly Frank"

#and delete pukeys of everyone who's Unfriendly
ccr -x Unfri

#create hashfile from a large file
ccr -sS hashfile.ccr < big_data.iso

#Check the hashfile
ccr -vS hashfile.ccr < the_same_big_data.iso

#create (ascii-armored) symmetric key and encrypt a large file
ccr -g sha256,chacha20 -aS symkey.asc
ccr -eaS symkey.asc -R big_data.iso -o big_data_encrypted.iso

#decrypt a large file
ccr -daS symkey.asc <big_data_encrypted.iso >big_data.iso

#password-protect all your private keys
ccr -L

#protect a symmetric key using another symmetric key
ccr -L -S symkey1 -w symkey2
```

```
#password-protect symkey2 with a custom cipher  
ccr -L -S symkey2 -w @xsynd,cube512
```

To see details of commands, see the following links:

<https://github.com/exaexa/codecrypt>

<https://e-x-a.org/codecrypt>

**User Manual (Command Line) CODECRYPT**

<https://e-x-a.org/codecrypt/ccr.1.html>

## 12. Annex "OpenQbit Quantum Computing".

### How does quantum computing work?

The digital transformation is bringing about change in the world faster than ever before. Would you believe that the digital era is about to end? **Digital literacy** has already been identified as an area where open knowledge and accessible opportunities to learn about technology are urgent to address gaps in social and economic development. Learning from the key concepts of the digital age will become even more critical with the imminent arrival of another new technological wave capable of transforming existing models with amazing speed and power: **quantum technologies**.

In this article, we compare the basic concepts of traditional computing and quantum computing; and we also begin to explore their application in other related areas.

What are quantum technologies?

Throughout history, human beings have developed technology as they have understood how nature works through science. Between 1900 and 1930, the study of some physical phenomena that were not yet well understood gave rise to a new physical theory, **Quantum Mechanics**. This theory describes and explains the functioning of the microscopic world, the natural habitat of molecules, atoms or electrons. Thanks to this theory, not only has it been possible to explain these phenomena, but it has also been possible to understand that subatomic reality works in a completely counter-intuitive, almost magical way, and that in the microscopic world events take place that do not occur in the macroscopic world.

These quantum **properties** include quantum superposition, quantum entanglement and quantum teleportation.

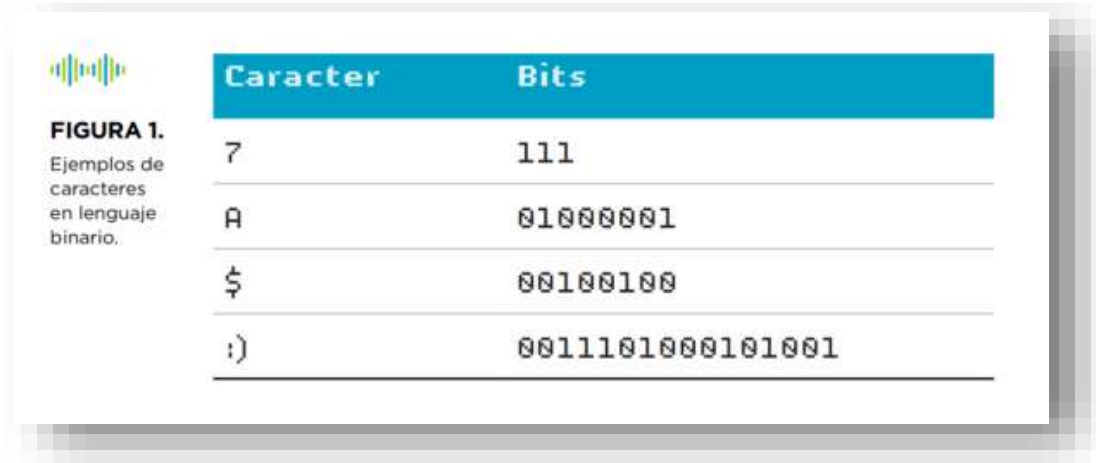
- **Quantum superposition** describes how a particle can be in different states at the same time.
- **Quantum entanglement** describes how two particles as far apart as desired can be correlated in such a way that, when interacting with one, the other is aware of it.
- **Quantum teleportation** uses quantum entanglement to send information from one place to another in space without having to travel through it.

Quantum technologies are based on these quantum properties of the subatomic nature.

In this case, today the understanding of the microscopic world through Quantum Mechanics allows us to invent and design technologies capable of improving people's lives. There are many and very different technologies that use quantum phenomena and some of them, such as lasers or magnetic resonance imaging (MRI), have been with us for more than half a century. However, we are currently witnessing a technological revolution in areas such as

quantum computing, quantum information, quantum simulation, quantum optics, quantum metrology, quantum clocks or quantum sensors.

What is quantum computing? First, you have to understand classical computing.



**FIGURA 1.**  
Ejemplos de caracteres en lenguaje binario.

Caracter	Bits
7	111
A	01000001
\$	00100100
:)	0011101000101001

To

understand how quantum computers work, it is convenient to first explain how the computers we use every day, which we will refer to in this document as digital or classic computers, work. These, like the rest of the electronic devices such as tablets or mobile phones, use bits as the fundamental units of memory. This means that programs and applications are encoded in bits, that is, in binary language of zeros and ones. Every time we interact with any of these devices, for example, by pressing a key on the keyboard, strings of zeros and ones are created, destroyed and/or modified inside the computer.

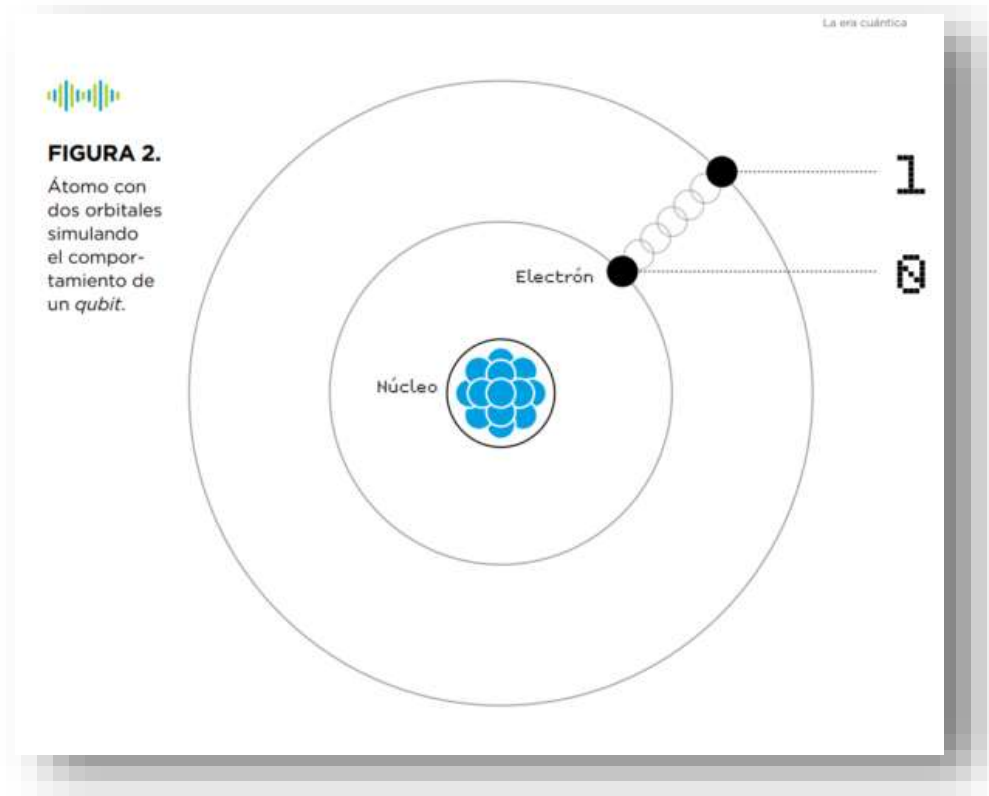
The interesting question is, what are these zeros and ones physically inside the computer? The zero and one states correspond to electrical current that circulates, or not, through microscopic parts called transistors, which act as switches. When no current is flowing, the transistor is "off" and corresponds to bit 0, and when it is flowing it is "on" and corresponds to bit 1.

More simply, it is as if bits 0 and 1 correspond to holes, so that an empty hole is a bit 0 and a hole occupied by an electron is a bit 1. This is why these devices are called electronics. As an example, figure 1 shows the binary writing of some characters. Now that we have an idea of how today's computers work, let's try to understand how quantum work.

### From bits to qubits

The fundamental unit of information in quantum computing is the quantum bit or qubit. Qubits are, by definition, two-level quantum systems -we will see examples here- which, like bits, can be at the low level, which corresponds to a state of low excitation or energy defined as 0, or at the high level, which corresponds to a state of higher excitation or defined as 1. However, and here lies the fundamental difference with classical computing, qubits can also

be in any of the infinite intermediate states between 0 and 1, such as a state that is half 0 and half 1, or three quarters of 0 and a quarter of 1.



Quantum algorithms, exponentially more powerful and efficient computing

The purpose of quantum computers is to take advantage of these quantum properties of *qubits*, as quantum systems that they are, in order to run quantum algorithms that use overlapping and interleaving to provide much greater processing power than the classics. It is important to point out that the real change of paradigm does not consist in doing the same thing that digital or classical computers do -the current ones- but faster, as it can be read in many articles, but that quantum algorithms allow to perform certain operations in a totally different way that in many cases turns out to be more efficient -that is, in much less time or using much less computational resources-.

Let's look at a concrete example of what this involves. Let's imagine that we are in Bogotá and we want to know the best route to get to Lima from among a million options to get there ( $N=1,000,000$ ). In order to use computers to find the optimal route we need to digitize 1,000,000 options, which implies translating them into bit language for the classic computer and into *qubits* for the quantum computer. While a classic computer would need to go one by one analyzing all the paths until finding the desired one, a quantum computer takes advantage of the process known as quantum parallelism that allows it to consider all the paths at once. This implies that, while the classical computer needs the order of  $N/2$  steps or

iterations, that is, 500,000 attempts, the quantum computer will find the optimal path after only  $\sqrt{N}$  operations on the registry, that is, 1,000 attempts.

In the previous case the advantage is quadratic, but in other cases it is even exponential, which means that with  $n$  *qubits* we can obtain a computational capacity equivalent to  $2^n$  bits. To exemplify this, it is common to count that with about 270 qubits we could have more base states in a quantum computer - more different and simultaneous character strings - than the number of atoms in the universe, which is estimated to be around  $10^{80}$ . Another example is that it is estimated that with a quantum computer of between 2000 and 2500 *qubits* we could break practically all the cryptography used today (the so-called public key cryptography).

Why is it important to know about quantum technology?

We are in a moment of digital transformation in which different emerging technologies such as blockchain, artificial intelligence, drones, Internet of things, virtual reality, 5G, 3D printers, robots or autonomous vehicles have more and more presence in multiple fields and sectors. These technologies, called to improve the quality of life of the human being accelerating the development and generating social impact, advance nowadays in a parallel way. Only rarely do we see companies developing products that exploit combinations of two or more of these technologies, such as blockchain and IoT or drones and artificial intelligence. Although they are destined to converge, thus generating an exponentially greater impact, the initial stage of development in which they find themselves and the scarcity of developers and people with technical profiles mean that convergence is still a pending task.

Because of their disruptive potential, quantum technologies are expected not only to converge with all these new technologies, but to have a cross-cutting influence on virtually all of them. Quantum computing will threaten authentication, exchange and secure storage of data, having a major impact on those technologies where cryptography has a more relevant role, such as cyber security or blockchain, and a minor negative impact but also to be considered in technologies such as 5G, IoT or drones.

Do you want to practice quantum computing?

Dozens of quantum computer simulators are already available on the net with different programming languages already in use such as C, C++, Java, Matlab, Maxima, Python or Octave. Also, new languages like Q#, launched by Microsoft. You can explore and play with a virtual quantum machine through platforms such as IBM and Rigetti.

Mini QRNG is created by OpenQbit.com company that focuses on developing quantum computing based technology for different types of sectors both private and public.

**Why Mini QRNG is different from other QRNGs, simply because the system was created to be modular and can be easily assembled at home at a fairly low cost.**

- (1) <https://blogs.iadb.org/conocimiento-abierto/es/como-funciona-la-computacion-cuantica/>

### 13. Licensing and use of software.

Android

<https://source.android.com/setup/start/licenses>

Termux

<https://github.com/termux/termux-app/blob/master/LICENSE.md>

Node

<https://raw.githubusercontent.com/nodejs/node/master/LICENSE>

Python

<https://www.python.org/download/releases/2.7/license/>

OpenSSH

<https://www.openssh.com/features.html>

Putty SSH

<https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html>

MIT App Inventor 2 Companion and App Inventor Blockly

<https://appinventor.mit.edu/about/termservice>

External extensions:



## JSOONTOOLS

<https://thunkableblocks.blogspot.com/2017/07/jsontools-extension.html>

Licensing opensource and commercial versions of the QRNG Mini system see the official website <http://www.openqbit.com>

Mini QRNG, Mini BlocklyChain, MiniBlockly, BlocklyCode, MiniBlockMiniChain, QBlockly, MiniPQC son marcas registradas por OpenQbit.

Mini PQC (Post-Quantum Cryptography) is in the public domain.

All code and documentation in Mini PQC has been dedicated to the public domain by the authors. All code authors and representatives of the companies they work for have signed affidavits dedicating their contributions to the public domain and the originals of those affidavits are stored in a safe at OpenQbit Mexico's main offices. Any person is free to publish, use or distribute the original Mini PQC (OpenQbit) extensions, either as source code or as compiled binaries, for any purpose, commercial or non-commercial, and by any means.

The previous paragraph applies to the code and documentation deliverable in Mini PQC those parts of the Mini PQC library that you actually group and ship with a larger application. Some scripts used as part of the compilation process (for example, "configuration" scripts generated by autoconf) may be included in other open source licenses. However, none of these compilation scripts make it into the final Mini PQC deliverable library, so the licenses associated with those scripts should not be a factor in evaluating your rights to copy and use the Mini PQC library.

All the deliverable code in Mini PQC has been written from scratch. No code has been taken from other projects or from the open internet. Each line of code can be traced back to its original author, and all those authors have public domain dedications on file. Therefore, the Mini PQC code base is clean and not contaminated with code licensed from other open source projects, not open contribution

Mini QRNG is open source, which means you can make as many copies as you want and do what you want with those copies, without limitation. But Mini PQC is not open source. To keep Mini PQC in the public domain and ensure that the code is not contaminated with proprietary or licensed content, the project does not accept patches from unknown people. All code in Mini PQC is original, as it has been written specifically for use by Mini PQC. No code has been copied from unknown sources on the Internet.

Mini PQC is in the public domain and does not require a license. Still, some organizations want legal proof of their right to use Mini PQC. Circumstances where this occurs include the following:

- Your company wants compensation for claims of copyright infringement.
- You are using Mini PQC in a jurisdiction that does not recognize the public domain.
- You are using Mini PQC in a jurisdiction that does not recognize an author's right to place his or her work in the public domain.
- You want to have a tangible legal document as evidence that you have the legal right to use and distribute Mini PQC.
- Your legal department tells you that you must buy a license.

If any of the above circumstances apply to you, OpenQbit, the company that employs all Mini PQC developers, will sell you a Mini PQC Title Guarantee. A Title Warranty is a legal document that states that the claimed authors of Mini PQC are the true authors, and that the authors have the legal right to dedicate the Mini PQC to the public domain, and that OpenQbit will vigorously defend itself against licensing claims. All proceeds from the sale of Mini PQC's title warranties are used to fund the continuous improvement and support of Mini PQC.

#### Contributed Code

To keep Mini PQC completely free and royalty-free, the project does not accept patches. If you would like to make a suggested change and include a patch as a proof of concept, that would be great. However, don't be offended if we rewrite your patch from scratch. The type of non-commercial or opensource license who uses it in this modality and some similar without purchase of support either individual or corporate use no matter the size of the company will be governed by the following legal premises.

Warranty disclaimer. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) "AS IS", **WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either** express or implied, including, without limitation, any warranties or conditions of TITLE, NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the correct use or redistribution of the Work and for assuming any risks associated with your exercise of permissions under this License.

Any financial or other losses incurred by the use of this software will be borne by the affected party. All legal disputes the parties will submit to courts only in the jurisdiction of Mexico City, country Mexico.

For commercial support, use and licensing an agreement or contract must be established between OpenQbit or its corporate and the interested party.

The terms and conditions of distribution marketing may change without notice, please go to the official website [www.openqbit.com](http://www.openqbit.com) to see any changes to support and licensing clauses non-commercial and commercial.

Any person, user, private or public entity of any legal nature or from any part of the world who simply uses the software accepts without conditions the clauses established in this document and those that can be modified at any time in the portal of [www.openqbit.com](http://www.openqbit.com) without previous notice and can be applied at the discretion of OpenQbit in non-commercial or commercial use.

Any questions and information about Mini PQC should be directed to the App Inventor community or to the various Blockly system communities as they are: AppBuilder, Trunkable, etc. and / or to the mail [opensource@openqbit.com](mailto:opensource@openqbit.com) for the demand of questions can take 3 to 5 working days to answer.

Support with commercial use.

[support@openqbit.com](mailto:support@openqbit.com)

Sales for commercial use.

[sales@openqbit.com](mailto:sales@openqbit.com)

Legal information and licensing questions or concerns

[legal@openqbit.com](mailto:legal@openqbit.com)

