



Instalación, configuración & administración.

Post-Quantum Cryptography (PQC)

Manual usuario

versión 1.0 Beta

Agosto 2020.

MiniPQC es una marca registrada de OpenQbit Inc, bajo licencia de uso libre y comercial.
Términos y condiciones de uso en: www.OpenQbit.com

Contenido

1. Introducción.....	3
2. ¿Qué es la programación Blockly?	4
3. ¿Qué es Termux?.....	4
4. ¿Qué es Mini PQC?	4
5. Configuración de almacenamiento “storage” dentro de Termux.....	8
6. Instalación de servidor SSH (Secure Shell).	9
7. Configuración de servidor SSH en teléfono móvil (smartphone).....	10
8. Instalacion del software algoritmos PQC (Post-Quantum Crytography).	16
9. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).	27
10. Definición y uso de bloques en Mini PQC.	28
11. Anexo “Programas Python”	42
12. Anexo “Computación Cuántica con OpenQbit”	53
13. Licenciamiento y uso de software.....	58

1. Introducción.

Históricamente hablando, la criptografía puede clasificarse en 4 etapas: clásica, moderna, cuántica y post-cuántica. El primer uso reconocido de la criptografía data de hace cerca de 4 mil años, en unos jeroglíficos no estándar encontrados en una tumba egipcia. La criptografía clásica, de una u otra manera continuó usándose conforme la civilización humana continuó su evolución.

La segunda guerra mundial marca un hito importante para la criptografía. Ante la necesidad de proteger los mensajes enviados a las tropas, aviones y submarinos a distancia, se realizaron grandes avances tanto en la construcción de máquinas criptográficas.

La siguiente gran revolución aparece a finales de los años 60 con el desarrollo de la criptografía asimétrica donde se utiliza una llave conocida por el público para cifrar los mensajes y otra llave privada para descifrarlos.

En los años 70's se tienen las primeras ideas relacionadas con la criptografía cuántica, destacando los algoritmos de Shor y Grover. Sin embargo, fue hasta los 80's cuando se muestran las primeras publicaciones de nuevos protocolos que basaban su seguridad en los principios de la mecánica cuántica —como el de incertidumbre o el de superposición— utilizando láseres para emitir información en un fotón.

Cuando los algoritmos de la criptografía de llave pública empiezan a verse comprometidos por el cómputo cuántico, surge la criptografía post-cuántica, que hace referencia a los algoritmos diseñados para resistir ataques de computadoras cuánticas. En ella se aprecian varias ramas que se diferencian por la forma en cómo operan, siendo estas: algoritmos basados en rejillas, algoritmos basados en ecuaciones multivariadas, algoritmos basados en curvas elípticas isógenas supersingulares, algoritmos basados en hashes, y algoritmos basados en códigos.

Desde el punto de vista del uso y desempeño de este tipo de algoritmos, se ha explorado su ejecución en sistemas con arquitectura ARM básicamente orientados a teléfonos móviles como lo son: Android, Arduino y Raspberry Pi. De ahí que los antecedentes relacionados con este trabajo exhiben trabajos con algoritmos post-cuánticos que podrían dividirse en: ejecuciones en dispositivos basados en ARM y sistema operativo Android, en dispositivos de uso específico y trabajos de algoritmos post-cuánticos en la arquitectura ARM, los cuales radican en cifrado, firmas digitales y acuerdos de llave, dejando abierta la posibilidad de explorar la primitiva de hash, algoritmos como **CHACHA20**, **NTRU**, **NEWHOPE**, **SALSA20**, **SPHINCS+**, **RAIBOW** y **CODECRYPT**, enfocada al servicio de integridad, así como su comportamiento en sistemas con recursos limitados.

2. ¿Qué es la programación Blockly?

Blockly es un **lenguaje de programación visual** compuesto por un sencillo conjunto de comandos que podemos combinar como si fueran las piezas de un rompecabezas. Es una herramienta muy útil para el que quiera **aprender a programar** de una forma intuitiva y simple o para quien ya sabe programar y quiera ver el potencial de este tipo de programación.

Blockly es una forma de programación en donde no se necesita algún antecedente de ningún tipo de lenguaje de computación o informática, esto se debe a que únicamente es unir bloques gráficos como si estuviéramos jugando lego o un rompecabezas, solo se necesita tener un poco de lógica y ¡Listo!!!

Cualquiera puede crear programas para teléfonos móviles (smartphones) sin meterse con esos lenguajes de programación difíciles de entender, solo arma bloques de forma gráfica de forma sencilla, fácil y rápida de crear.

3. ¿Qué es Termux?

Termux es un emulador de terminal Android y una aplicación de entorno Linux que funciona directamente sin necesidad de enrutamiento o configuración. Un sistema base mínimo se instala automáticamente.

Usaremos Termux por su estabilidad y fácil instalación y manejo, sin embargo, se puede usar un ambiente instalado de Linux Ubuntu para Android.

En este ambiente de Linux tendrá el “core” de los procesos de los algoritmos de Post Cuanticos Criptograficos del MiniPQC.

4. ¿Qué es Mini PQC?

Mini PQC (Mini Post-Quantum Cryptography) es Software que incluye las siguientes soluciones tecnológicas (algoritmos) para poder crear PQC (Post-Quantum Cryptography) como sigue:

- CHACHA20
- NEWHOPE
- NTRU
- SALSA20
- SPHINCS+
- RAIBOW
- CODECRYPT

1. Instalación y configuración de Terminal Termux.

Necesitamos primero un ambiente Linux ya que todo sistema Android está basado en Linux por seguridad y flexibilidad en herramientas, usaremos la terminal “Termux” que contiene dicho ambiente en donde instalaremos la(s) herramienta(s) que nos ayudaran para la creación de QRNGs.

Termux es un emulador de Linux donde instalaremos los paquetes necesarios para crear de números cuanticos.

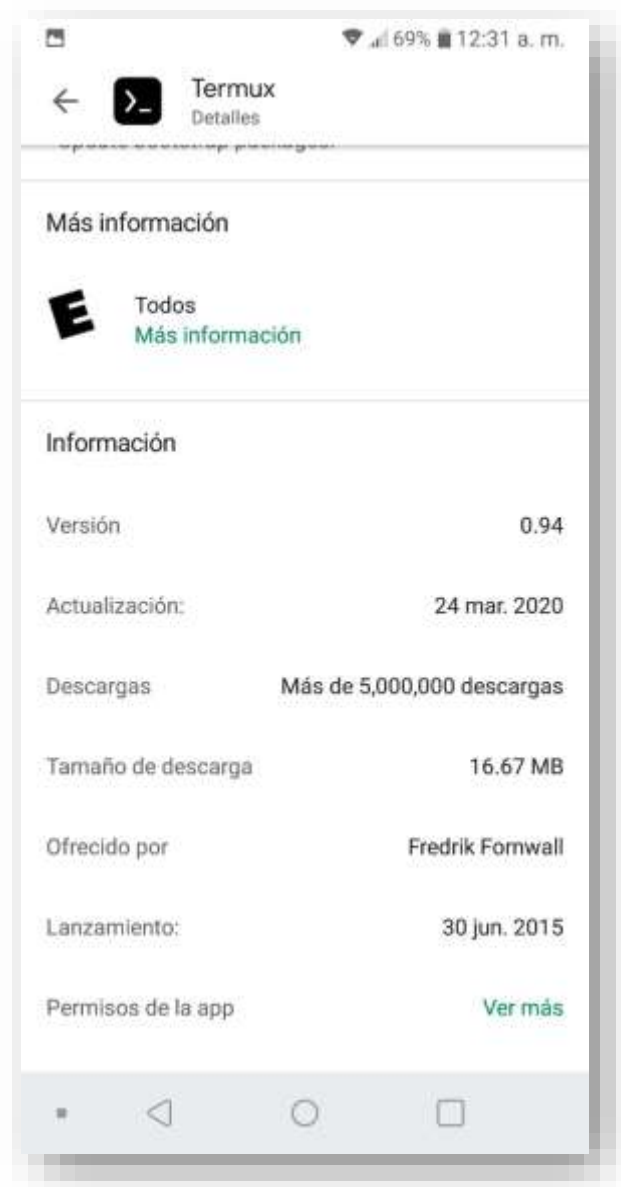
Una de las principales ventajas al usar Termux es que podrás instalar programas sin tener que “rootear” el móvil (Smartphone) esto asegura que por esta instalación no se pierde ningún tipo de garantía del fabricante.

Instalación de Termux.

Desde tu móvil ve a la aplicación icono de Google Play (play.google.com).



Busca por aplicación “Termux”, selecciónala e iniciar el proceso de instalación.



Inicio de la aplicación Termux.

Después de iniciar tendremos que ejecutar los siguientes dos comandos para realizar actualizaciones del emulador del sistema operativo Linux:

\$ apt update

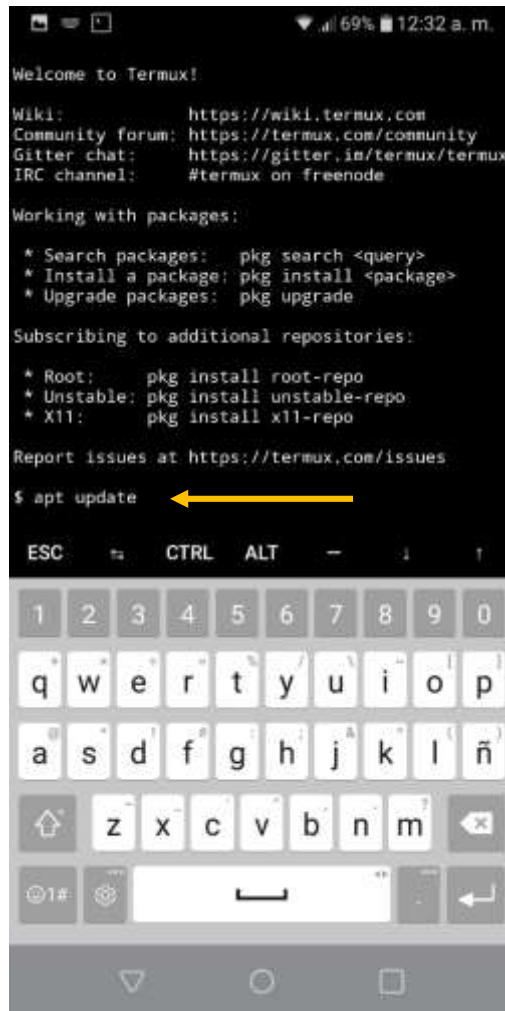
\$ apt upgrade

Confirmar todas las opciones Y(Yes)....

Inicio de Termux

\$ apt update

\$ apt upgrade



5. Configuración de almacenamiento “storage” dentro de Termux.

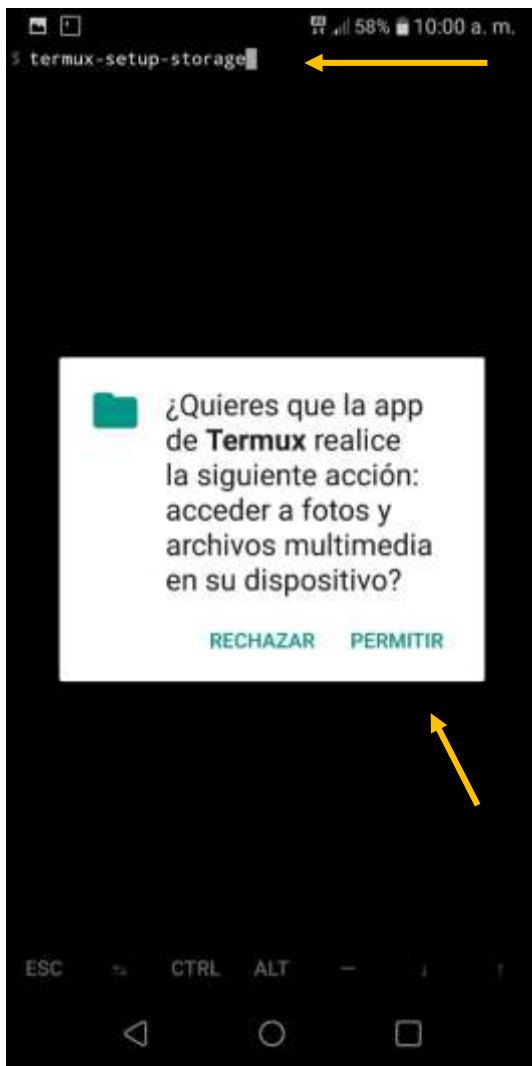
Después de haber realizado el update y upgrade del sistema Termux, empezaremos con la configuración de como poder ver el almacenamiento interno del teléfono en el sistema de Termux esto os ayudara a poder realizar intercambio de información entre Termux y nuestra información que tenemos en el teléfono.

Esto lo podemos realizar de una forma sencilla y rápida ejecutando el siguiente comando en una terminal de Termux.

\$ termux-setup-storage

Al ejecutar el anterior comando nos aparece una ventana pidiendo la confirmación de la creación de un **storage** virtual (directorio) en Termux damos click botón “PERMITIR” y creara el enlace al almacenamiento del teléfono. Verificamos dando el comando:

\$ ls



6. Instalación de servidor SSH (Secure Shell).

\$ apt install openssh

\$ apt install sshpass

\$ apt install openssh



```
$ 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



```
$ 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]
```

Hemos terminado con la instalación de la red de comunicaciones para servidor SSH localhost en móvil Smartphone.

7. Configuración de servidor SSH en teléfono móvil (smartphone).

Habilitaremos el servidor de SSH en el teléfono móvil para poder conectarnos desde nuestra PC al móvil y poder trabajar de una forma más rápida y cómoda, así mismo nos servirá para comprobar que el servicio del servidor SSH en el móvil funciona correctamente ya que este lo utilizaremos en la comunicación con Mini QRNG.

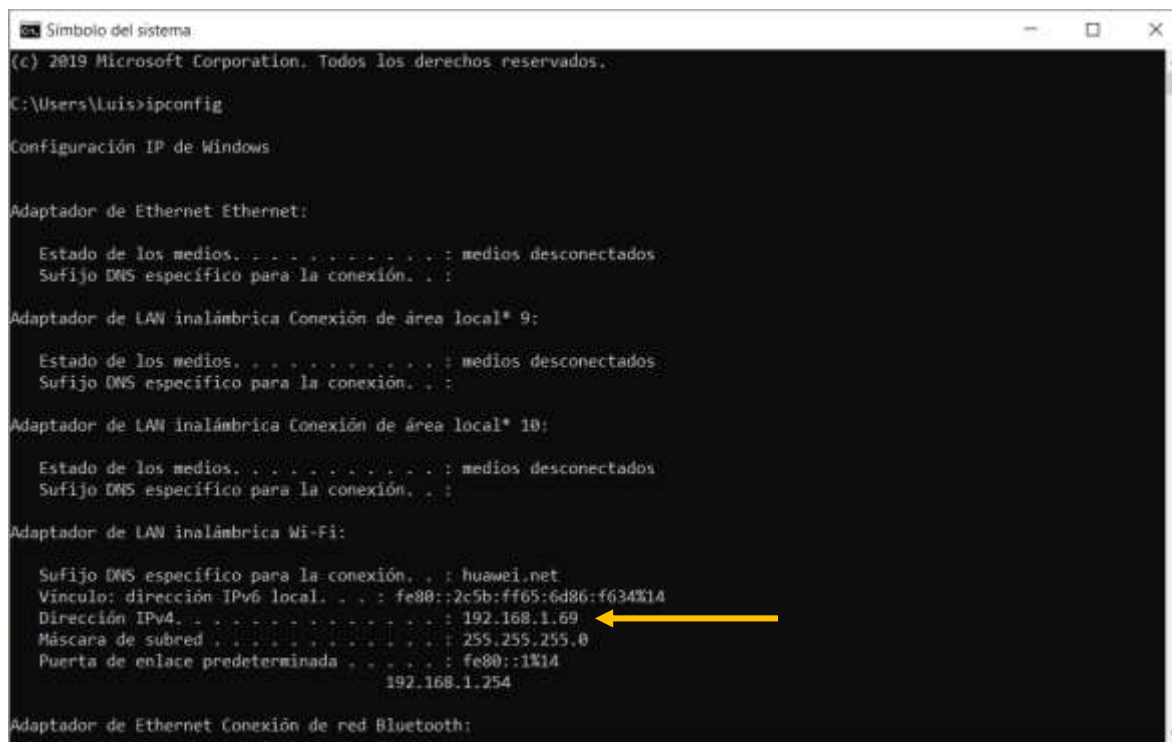
Lo primero que tenemos que hacer es conectar **a la misma red WiFi** el móvil y la PC para que se puedan ver. Las IPs o direcciones deben ser similares a 192.168.XXX.XXX los valores XXX son números variables que se asignan aleatoriamente en cada equipo.

Este ejemplo se probó en un móvil LG Q6 y una PC con Windows 10 Home.

Revisar la IP o dirección que tiene la PC conectada al WiFi deberemos abrir una terminal en Windows.

En el panel inferior donde está la lupa de buscar escribir cmd y presionar la tecla Enter. Se abrirá una terminal y en esta escribimos el comando:

```
C:\Users\nombre_usuario> 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:
```

Nos mostrara la IP que tiene asignada la PC en este es la 192.168.1.69 sin embargo esta los más probable es que sea diferente en cada caso.

NOTA: debe tomarse la dirección donde dice “Dirección IPv4” no confundir con la Puerta de enlace.

Ahora en el caso del teléfono móvil en la terminal de Termux debemos teclear el siguiente comando para saber cómo se llama nuestro usuario que usaremos para conectarnos al servidor SSH que tiene nuestro teléfono, ejecutamos el siguiente comando:

\$ whoami

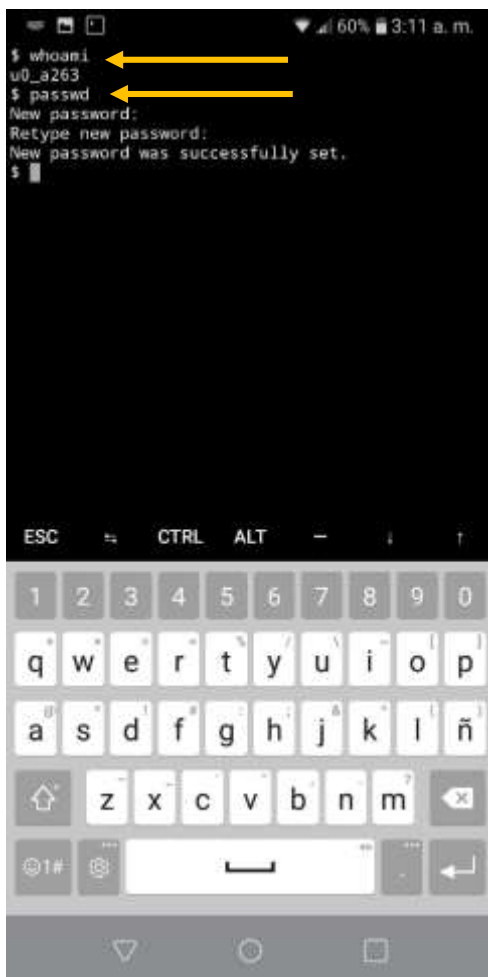
Posteriormente debemos darle un password a este usuario por lo que tenemos que ejecutar el siguiente comando:

\$ passwd


Nos pedirá que tecleemos un password y damos Enter, nuevamente nos pide el password confirmamos le damos el mismo y damos Enter, si ha sido exitosamente **"New password was successfully set"** en caso de marcar un error es posible que el password no se haya tecleado correctamente. Volver a realizar el procedimiento.

Y después para saber que IP tenemos en Termux tecleamos el siguiente comando, la IP esta después de la palabra **"inet"**:

\$ ifconfig -a



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



```
e 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 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 prefixlen 64 scopeid 0x20<link>
    unspec 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 frame 0
    TX packets 601034 bytes 93496881 (89.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

$
```

Ahora es momento de iniciar el servicio del servidor de SSH en el teléfono para que pueda recibir sesiones desde la PC. Ejecutamos el siguiente comando en la terminal de Termux, este comando no arroja ningún resultado.

\$ sshd



Ahora tendremos que instalar un programa en la PC que se comunicara con el servidor SSH del teléfono desde la PC.

Tenemos que ir a la página <https://www.putty.org>

Seleccionar donde se encuentra el enlace “You can download PuTTY here”



Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the 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 a



Bitvise SSH Client

Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported prof 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](#).

Escoger la versión de 32 bit, no importa si tu sistema es de 64 bits funcionara bien.

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. To see if the problem has already been fixed in those versions, see the [development snapshots](#).

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](#).)

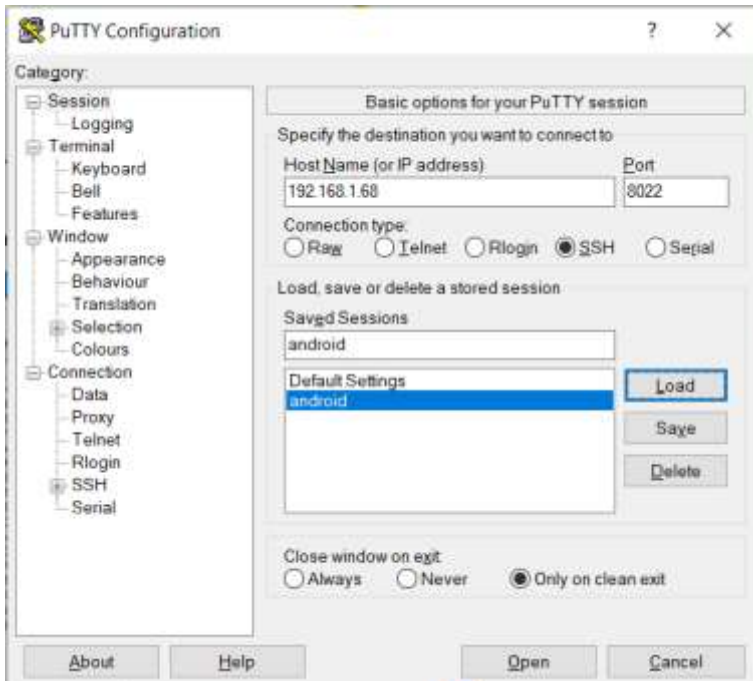
MSI ('Windows Installer')

32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	-------------	-------------

Ya que se haya bajado en tu PC ejecútalo e instalado con las opciones por default. Después inicia la aplicación de PuTTY.



En esta sesión introduciremos los datos de nuestro servidor Openssh que instalamos en el teléfono móvil.

Introducir la IP del teléfono móvil.

HostName or IP address:

192.168.1.68 (ejemplo IP)

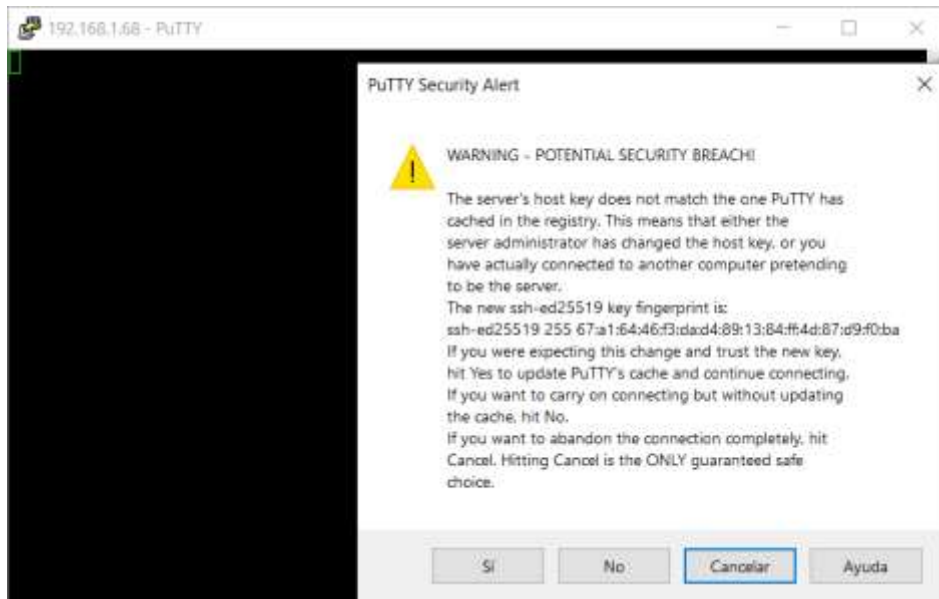
Port:

8022 (Puerto por default del servidor SSH del móvil).

Podemos dar un nombre de la sesión en "Saved Sessions" y damos click en el botón Save.

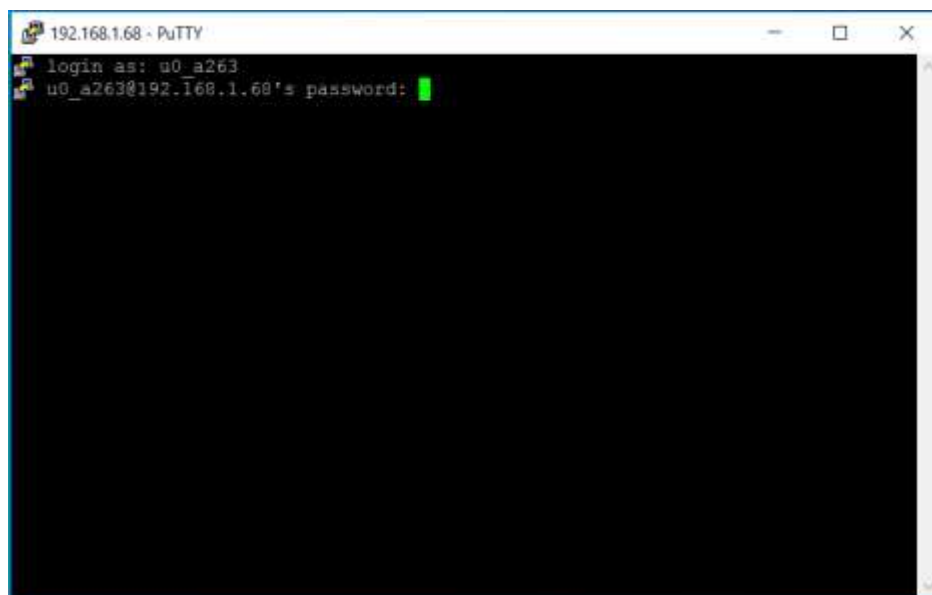
Posteriormente en la parte inferior presionamos para abrir una conexión al servidor dando el botón “Open”.

En la PC al conectarse por primera vez nos **pedirá por única vez** que confirmen la llave de cifrado de información le damos confirmación en el botón de “Si”.

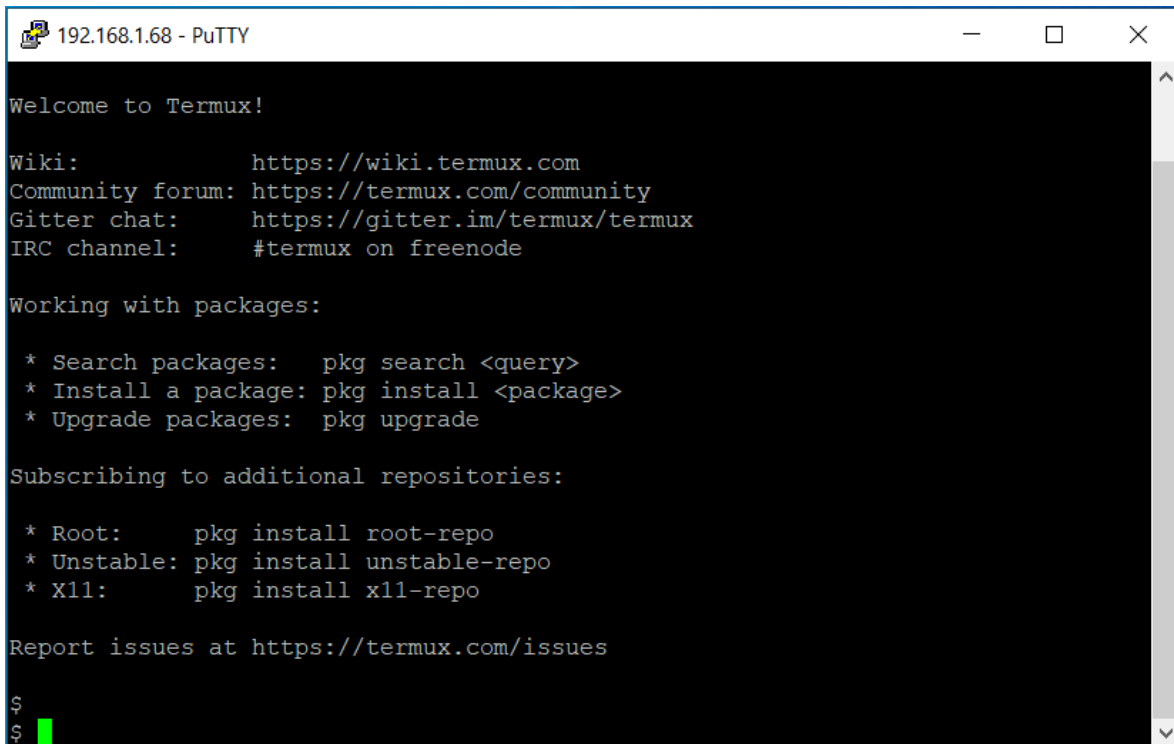


Posteriormente nos pedirá el usuario con el que nos vamos a conectar. Utilizaremos la información que sacamos con anterioridad (usuario y password).

En el **Login as:** debemos introducir nuestro usuario y dar Enter, después nos pedirá el password le damos nuevamente el botón de Enter.



Si los datos fueron los correctos, estaremos en una sesión de SSH (Secure Shell) realizada desde la PC (Cliente) en el teléfono (Servidor SSH).



```
192.168.1.68 - PuTTY
Welcome to Termux!

Wiki:          https://wiki.termux.com
Community forum: https://termux.com/community
Gitter chat:   https://gitter.im/termux/termux
IRC channel:   #termux on freenode

Working with packages:

* Search packages:  pkg search <query>
* Install a package: pkg install <package>
* Upgrade packages: pkg upgrade

Subscribing to additional repositories:

* Root:    pkg install root-repo
* Unstable: pkg install unstable-repo
* X11:     pkg install x11-repo

Report issues at https://termux.com/issues

$
$
```

NOTA IMPORTANTE: Recordemos que la IP (dirección) de la PC y la IP (dirección) teléfono móvil conectados en la misma WiFi estarán cambiando probablemente cada vez que nos desconectemos y volvamos a conectar por lo que hay que volver a verificar que direcciones tienen cada dispositivo, esto nos asegura el éxito de la conexión entre dispositivos por medio del servidor de SSH del teléfono y PC (Cliente).

8. Instalacion del software algoritmos PQC (Post-Quantum Cryptography).

Para la instalación de los algoritmos PQC primero necesitamos instalar el paquete de Python después hacer un upgrade para tener la ultima versión de Python y posteriormente instalar el modulo de Numpy con los siguientes comandos en la terminal de Termux.

\$ apt install Python

\$ pip install --upgrade pip

```
$ apt install python
Reading package lists... Done
Building dependency tree
Reading state information... Done
python is already the newest version (3.8.3).
0 upgraded, 0 newly installed, 0 to remove and 0
not upgraded.
$
```

```
$ pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/pac
kages/43/84/23ed6a1796480a6f1a2d38f2802901d07826
6bda38388954d01d3f2e821d/pip-20.1.1-py2.py3-none
-any.whl (1.5MB)
    | 10kB 2.0
    | 20kB 471
    | 30kB 565
    | 40kB 581
    | 51kB 547
    | 61kB 592
    | 71kB 635
    | 81kB 683
    | 92kB 721
    | 102kB 75
    | 112kB 75
    | 122kB 75
    | 133kB 75
    | 143kB 75
    | 153kB 75
    | 163kB 75
    | 174kB 75
$
```

```
8kB/s
Installing collected packages: pip
  Found existing installation: pip 19.2.3
  Uninstalling pip-19.2.3:
    Successfully uninstalled pip-19.2.3
  Successfully installed pip-20.1.1
$
```

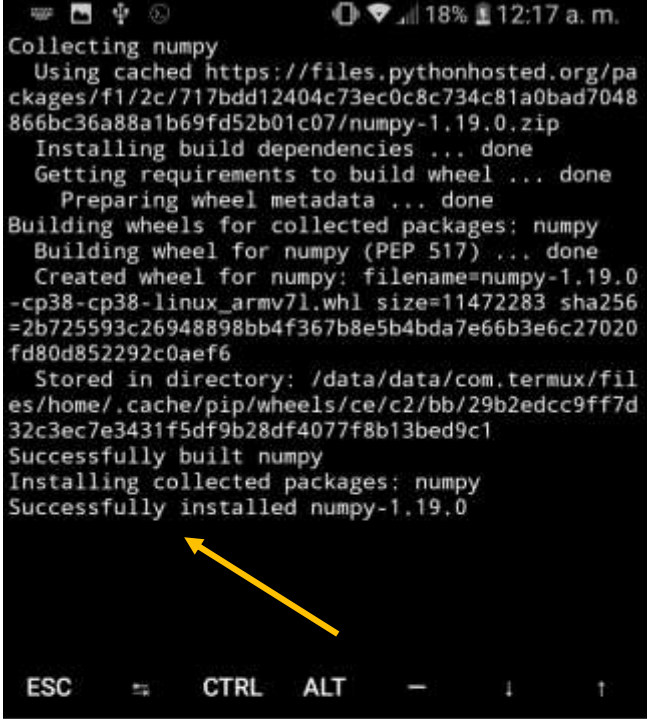

Instalacion modulo numpy con el siguiente comando:

\$ apt install numpy (Paciencia puede tardar la instalacion de **numpy** 40 minutos o mas)

\$ 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
```

Empecemos la instalacion de los algoritmos PQC (Post-Quantum Cryptography).

Implementacion de algoritmo Chacha20-POLY1305, es un código de autenticación de mensaje criptográfico creado por Daniel J. Bernstein. Se puede utilizar para verificar la integridad de los datos y la autenticidad de un mensaje. Una variante de Poly1305 de Bernstein que no requiere AES ha sido estandarizada por Internet Engineering Task Force en RFC 8439.

Instalacion de Chacha20-POLY1305 con el siguiente comando:

\$ apt install chacha20poly1305



```
$ pip install chacha20poly1305
Collecting chacha20poly1305
  Downloading chacha20poly1305-0.0.3.tar.gz (5.1 kB)
Building wheels for collected packages: chacha20poly1305
  Building wheel for chacha20poly1305 (setup.py)
    ... done
  Created wheel for chacha20poly1305: filename=chacha20poly1305-0.0.3-py2.py3-none-any.whl size=6682 sha256=a08824a3c9b1c3e1aef708f197793a60ef905c477820e7442f94e2532af35b81
  Stored in directory: /data/data/com.termux/files/home/.cache/pip/wheels/5c/a3/a0/0a27c3ae2fec0a4c94bac9685bcb83735d69fd08fbc837433d
Successfully built chacha20poly1305
Installing collected packages: chacha20poly1305
Successfully installed chacha20poly1305-0.0.3
$
```

Mas información de chacha20-Poly1305:

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

Implementación de algoritmo PQC **NTRU** este método acepta archivos de texto con mensajes cortos que no superen los 1 a 5 KBytes mas grandes puede tomar bastante tiempo en cifrar y descifrar archivos.

ntru es una implementación simple del sistema de cifrado NTRUEncrypt.

Las operaciones polinómicas se implementan utilizando la biblioteca SymPy. Se realizó en la Facultad de Matemáticas y Ciencias de la Información de la Universidad Tecnológica de Varsovia.

El sistema de criptografía de clave pública NTRU representa una mejora significativa en el mundo de la criptografía de clave pública: más fuerte y más pequeño que prácticamente cualquier otro sistema en uso y es resistente a la computadora cuántica.

\$ pip3 install sympy numpy docopt (numpy ya fue instalado en paginas anteriores)

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

The image consists of three terminal screenshots arranged horizontally, each showing a different step in the installation process. Each terminal has a status bar at the top showing battery level and time.

- Left Terminal:** Shows the command `$ pip3 install sympy`. The output indicates that `sympy-1.6.1-py3-none-any.whl` (5.8 MB) is being downloaded. A yellow arrow points to the command line.
- Middle Terminal:** Shows the command `$ pip3 install docopt`. The output shows `docopt-0.6.2.tar.gz` (25 kB) being downloaded and installed. A yellow arrow points to the command line.
- Right Terminal:** Shows the command `$ git clone http://github.com/jkrauze/ntru.git`. The output shows the repository being cloned into 'ntru'. A yellow arrow points to the command line.

En seguida generamos el par de llaves para usarls en el cifrado y descifrado de mensajes.

Creacion de llaves privada y publica (keypair) con el siguiente comando en la terminal Termux.

\$ 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**42 - 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

```

Mas información de NTRU: <https://ntru.org/>

Implementacion de algoritmo **NEWHOPE**.

NewHope es un protocolo de intercambio de claves basado en el problema Ring-Learning-with-errors (Ring-LWE), que se envió al proyecto cripto post-cuántico NIST.

Instalacion de NEWHOPE ejecutando el siguiente comando en la terminal Termux:

\$ 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
$
```

Mas informacion: <https://newhopecrypto.org/>

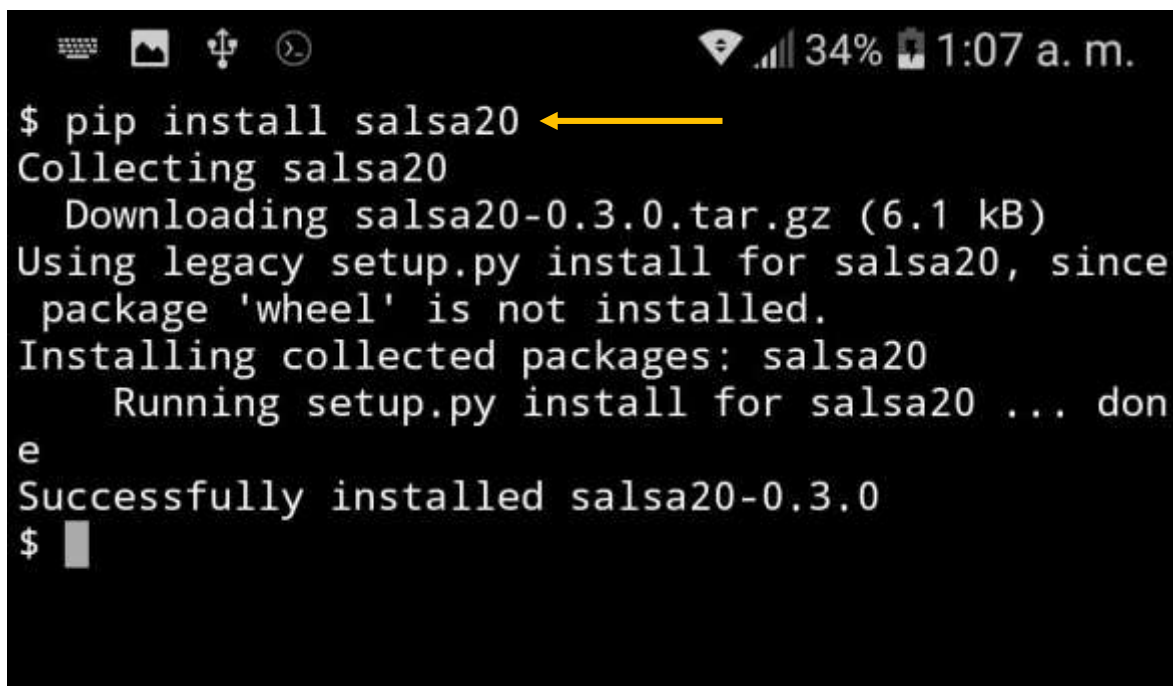
Implementación de algoritmo **SALSA20**.

Salsa20 y el ChaCha estrechamente relacionado son cifrados de flujo desarrollados por Daniel J. Bernstein. Salsa20, el cifrado original, fue diseñado en 2005, y luego Bernstein lo envió a eSTREAM. ChaCha es una modificación de Salsa20 publicada en 2008. Utiliza una nueva función redonda que aumenta la difusión y aumenta el rendimiento en algunas arquitecturas.

Ambos cifrados se basan en una función pseudoaleatoria basada en operaciones add-rotate-XOR (ARX): adición de 32 bits, adición de bits (XOR) y operaciones de rotación. La función principal asigna una clave de 256 bits, un nonce de 64 bits y un contador de 64 bits a un bloque de 512 bits del flujo de claves (también existe una versión Salsa con una clave de 128 bits). Esto le da a Salsa20 y ChaCha la ventaja inusual de que el usuario puede buscar eficientemente cualquier posición en la secuencia de claves en tiempo constante.

Instalación de Salsa20 ejecutando el siguiente comando:

\$ apt install salsa20

A screenshot of a terminal window with a black background and white text. The status bar at the top shows icons for keyboard, image, USB, and a clock, along with a Wi-Fi signal, cellular signal, 34% battery, and the time 1:07 a.m. The terminal text shows the command '\$ pip install salsa20' with a yellow arrow pointing to it. The output shows the package being collected, downloaded (6.1 kB), and installed using legacy setup.py, with a final confirmation 'Successfully installed salsa20-0.3.0'.

```
$ 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  
$
```

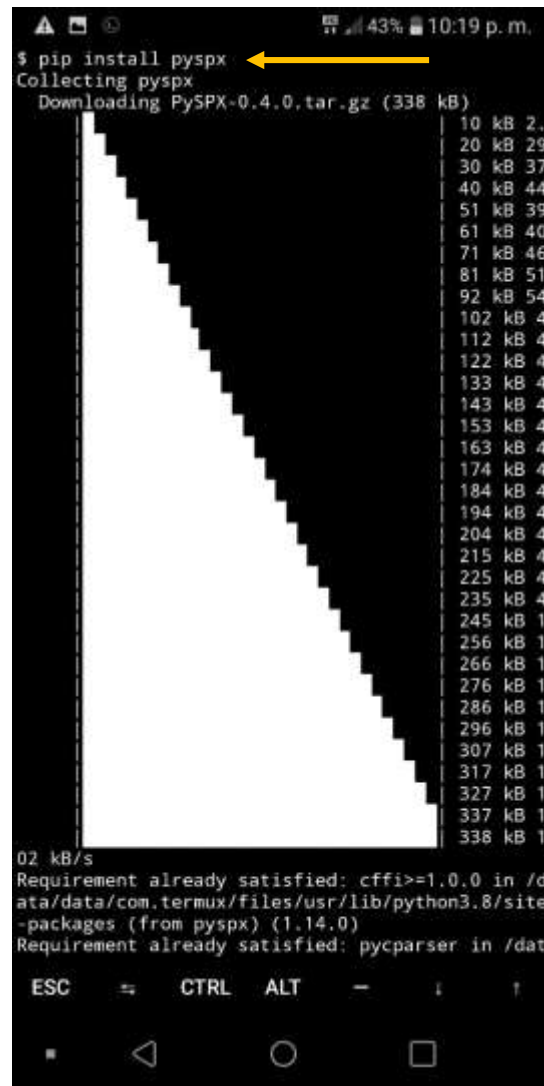
Más información: <https://cr.yp.to/snuffle/salsafamily-20071225.pdf>

Implementacion de algoritmo SPHINCS+

SPHINCS + es un esquema de firma basado en hash sin estado, que se envió al proyecto cripto post-cuántico NIST. El diseño avanza el esquema de firma SPHINCS, que se presentó en EUROCRYPT 2015. Incorpora múltiples mejoras, específicamente destinadas a reducir el tamaño de la firma. Para obtener una descripción general rápida de los cambios de SPHINCS a SPHINCS +.

Instalacion de SPHINCS+ ejecutando el siguiente comando:

\$ 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 /dat
  
```

Mas informacion: <https://sphincs.org/>

Implementación de algoritmo **RAINBOW**.

Una herramienta de generación de clave asimétrica resistente cuántica basada en el esquema de RAINBOW para firmas digitales.

Las firmas digitales son una forma común de autenticar a un remitente verificando una información adjunta al documento. En nuestro caso, la información adjunta es una secuencia de números generados usando el documento que se firmará.

Como se mencionó anteriormente, empleamos el esquema Generalized Unbalanced Oil and Vinegar o el esquema Rainbow para lograr lo mismo.

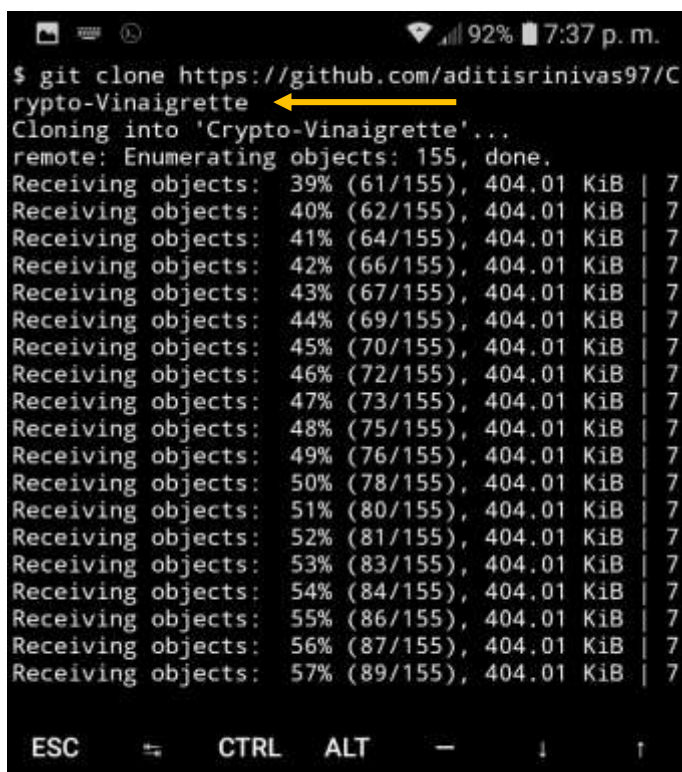
Antes de la instalación debemos tener instalado el paquete "git" con: `$ apt install git`

Instalación de RAINBOW ejecutando los siguientes comandos:

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

`$ cd Crypto-Vinaigrette`

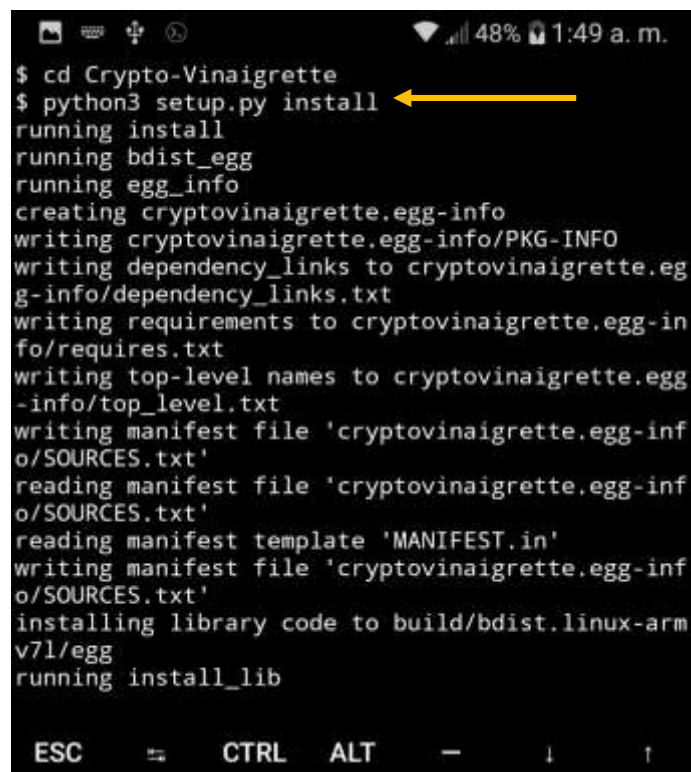
`$ python3 setup.py install`



```

$ 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

```



```

$ 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

```

Más información: <https://bit.ly/2vwckRw>

Implementacion de algoritmo **CODECRYPT**.

CODECRYPT. - Es una herramienta que engloba diferentes algoritmos PQC (Post-Quantum Cryptography) para realizar los principales procesos de cifrado y descifrado de texto y archivos.

Clasificados con los siguientes identificadores: [S]ign, [E]ncryption, [C]ipher y [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

Ahora realizaremos la instalación de la suite de herramientas CODECRYPT con la ejecución del siguiente comando en la Terminal termux, un punto importante es que con esta ejecución se instalaran *dos dependencias cryptopp y fftw*:

\$ 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) ...
$

```

9. Ambientes Blockly (App Inventor, AppyBuilder y Thunkable).

App Inventor es un entorno de desarrollo de software creado por Google Labs para la elaboración de aplicaciones destinadas al sistema operativo Android. El usuario puede, de forma visual y a partir de un conjunto de herramientas básicas, ir enlazando una serie de bloques para crear la aplicación. El sistema es gratuito y se puede descargar fácilmente de la web. Las aplicaciones creadas con App Inventor son muy fáciles de crear debido a que no se necesita conocimiento de ningún tipo de lenguaje de programación.

Todos los ambientes actuales que usan la tecnología de Blockly como son AppyBuilder y Thunkable entre otros tienen su versión gratuita, su forma de uso puede ser a través de internet en sus diversos sitios o también puede ser instalado en casa.

Los bloques que forman la arquitectura de Mini BloclChain han sido probados en App Inventor y AppyBuilder sin embargo por su optimización de código deberán funcionar en las otras plataformas.

Versiones de por internet:

App Inventor.

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

AppyBuilder.

<http://appybuilder.com/>

Thunkable.

<https://thunkable.com/>

Versión para ser instalada en tu equipo (PC):

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

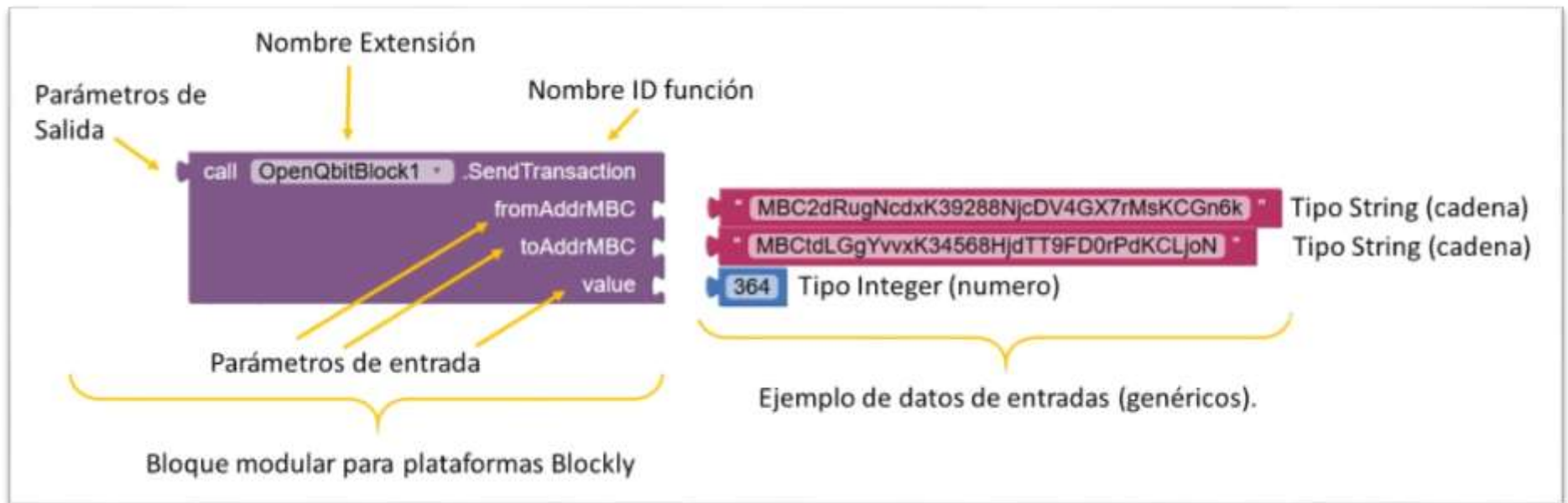
Ambiente para desarrolladores de bloques Blockly.

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

10. Definición y uso de bloques en Mini PQC.

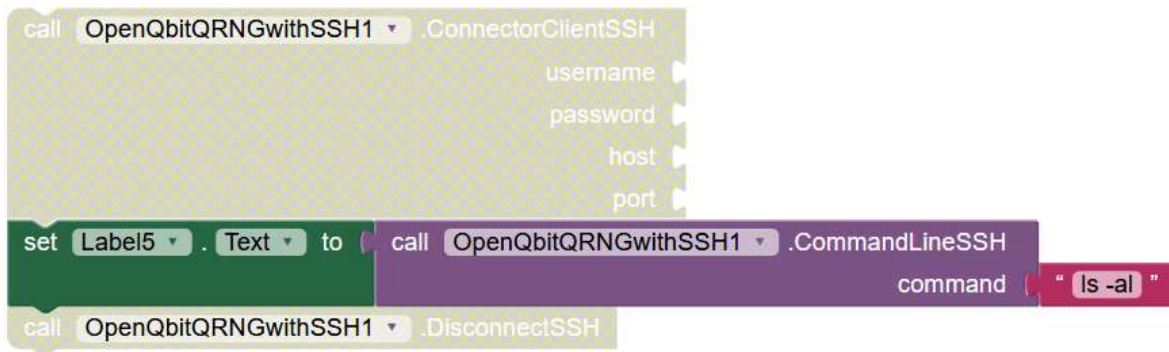
Empezaremos con explicar la distribución de los datos que tendrá todos los bloques, su sintaxis de uso y configuración.

En el siguiente ejemplo podemos ver un bloque modular y sus parámetros de entrada y salida, así como los tipos de datos de entrada, estos datos pueden ser de tipo String (cadena de caracteres) o Integer (numero entero o decimal). Mostramos como es su uso y configurarlo para su funcionamiento adecuado.



Cada bloque modular tendrá su descripción y se nombrará en caso de tener alguna(s) dependencia(s) obligatoria(s) u opcional(es) de otros bloques usados como parámetros de entrada se anunciará el proceso de integración. Comencemos con los bloques de la extensión OpenQbitPQCwithSSH.

Bloque para ejecutar comando en la terminal de Termux – (**CommandLineSSH**)



Parámetros de entrada: **command** <String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Ejecuta el comando introducido en a terminal Termux.

Descripción: Se ejecuta un comando introducido y se necesita antes el bloque para conectarse al servidor SSH (**ConnectorClientSSH**) y después usar el bloque (**DisconnectSSH**).

Bloque para conectarse a un servidor SSH remoto o local – (**ConnectorClientSSH**).

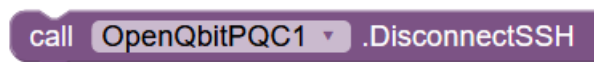


Parámetros de entrada: **username** <string>, **password** <string>, **host** <string>, **port**<integer>

Parámetros de salida: Si es exitosa la conexión con el servidor ssh de la terminal Termux nos entrega un mensaje; "**Connect SSH**", en caso de no ser exitosa nos entrega un mensaje **NULL**.

Descripción: Bloque de comunicación para conectar a servidor SSH elegido a la terminal Termux, vía protocolo de comunicación SSH (Secure Shell).

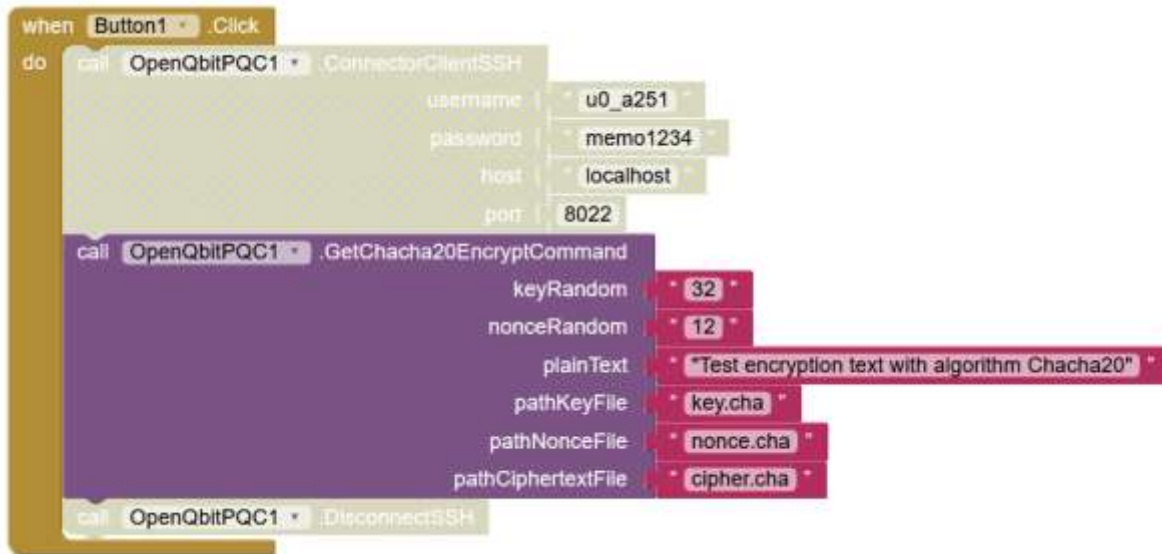
Bloque para cerrar la sesión SSH (**DisconnectSSH**).



Parámetros de entrada y salida: No aplica (ninguno).

Descripción: Bloque para cerrar sesión de SSH.

Bloque para cifrar cadenas de caracteres con algoritmo Post-Quantum Cryptography PQC Chacha20Poly1302 – (GetChacha20EncryptCommand)



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

Dependencia obligatoria: Bloque (ConnectorClientSSH), Bloque (DisconnectSSH).

Los parámetros keyRandom y nonceRandom son números enteros para la generación de números pseudoaleatorios, si deseas números aleatorios puedes usar la extensión (OpenQbitQRNGwithSSH), en nuestro ejemplo usamos pseudoaleatorios.

Parámetros de salida: Ejecuta el evento (OutputDataChacha20). Entrega una cadena de caracteres cifrada en base64.



Ejemplo: "Test encryption text with algorithm Chacha20"

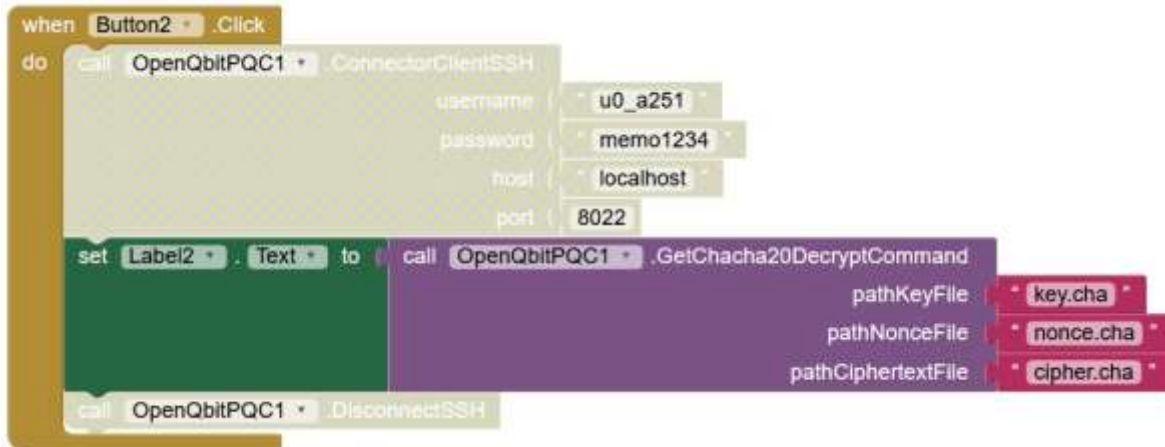
Output:

OCtxa3YlrpOUKRUn3tpcsjU7w0ZckO/4FMx0/Ybsz3hISOXNBY/GOB8Jf6FAVrwz135q+h6A4f/m
Ooi

Se generan tres archivos para usarlos en el proceso de descifrado en la ruta especificada. Los nombres pueden ser a selección del usuario, recordando que cada vez que se ejecute el bloque si no se cambia de nombre sera reescrito por el nuevo mensaje cifrado.

Descripción: Bloque que cifra una cadena de caracteres con el algoritmo Chacha20Poly1305 aplicando PCQ (Post-Quantum Cryptography).

Bloque para decifrar cadenas de caracteres con algoritmo Post-Quantum Cryptography PQC Chacha20Poly1302 – (**GetChacha20DecryptCommand**)



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

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Entrega la cadena descifrada que contiene el archivo en la ruta especificada **pathCiphertextFile** este archivo fue generado en el bloque de cifrado (**GetChacha20EncryptCommand**).

Ejemplo:

OCTxa3YlrpOUKRUN3tpcsjJU7w0ZckO/4FMx0/Ybsz3hISOXNBY/G0B8Jf6FAVrwz135q+h6A4f/m
Ooi

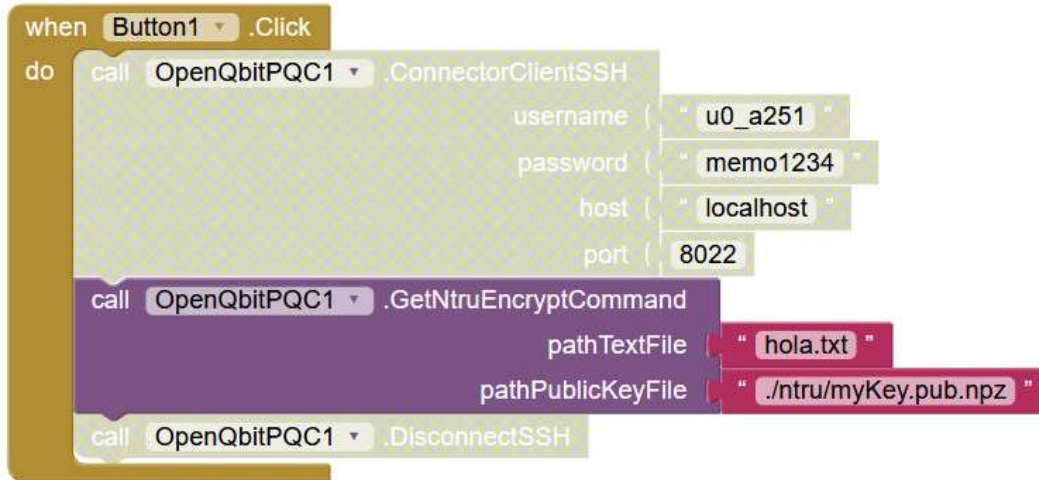
Output: "Test encryption text with algorithm Chacha20"

NOTA IMPORTANTE:

Los bloques anteriores (**GetChacha20EncryptCommand**) y (**GetChacha20DecryptCommand**) antes de usarlos deberá instalar en la Terminal de Termux los programas **chacha20encrypt.py** y **chacha20decrypt.py** y situarlos en el home de usuario por default de Termux que es: **/data/data/com.termux/files/home**, este usuario es con el que se están conectando via SSH (Secure Shell)

Ver Anexo "Programas Python".

Bloque para cifrar un archivo con texto plano con algoritmo Post-Quantum Cryptography PQC NTRU – (GetNtruEncryptCommand)



Parámetros de entrada: **pathTextFile**<String>, **pathPublicKeyFile**<String>

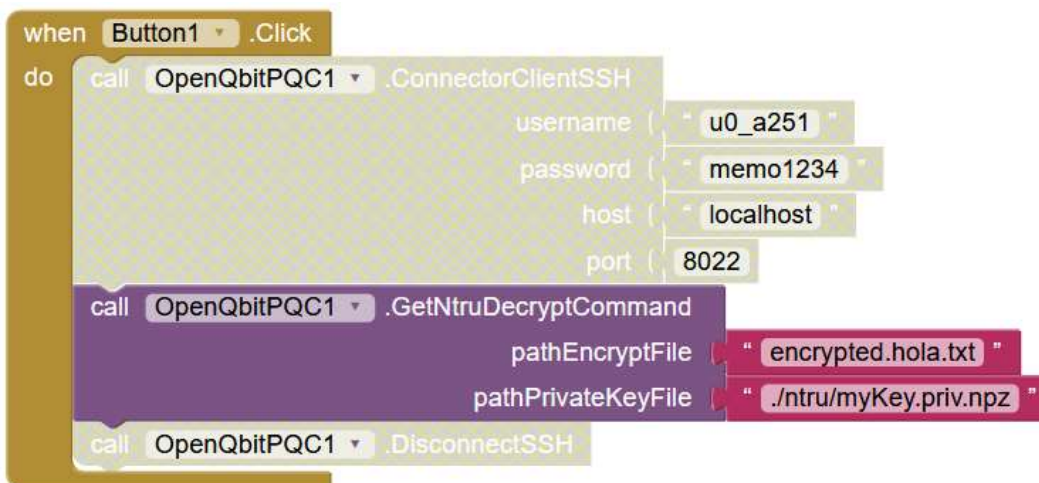
Dependencia obligatoria: Bloque (ConnectorClientSSH), Bloque (DisconnectSSH).

Parámetros de salida: Entrega un archivo cifrado con la palabra de inicio: **encrypted**

En nuestro caso se introdujo el archivo hola.txt el resultado seria un archivo con el nombre: **encrypted.hola.txt**

Descripción: Cifra archivos de texto pequeños .txt estos deben tener **máximo 100 caracteres**.

Bloque para decifrar un archivo de texto con algoritmo Post-Quantum Cryptography PQC NTRU – (GetNtruDecryptCommand)



Parámetros de entrada: **pathEncryptFile<String>**, **pathPrivateKeyFile<String>**

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

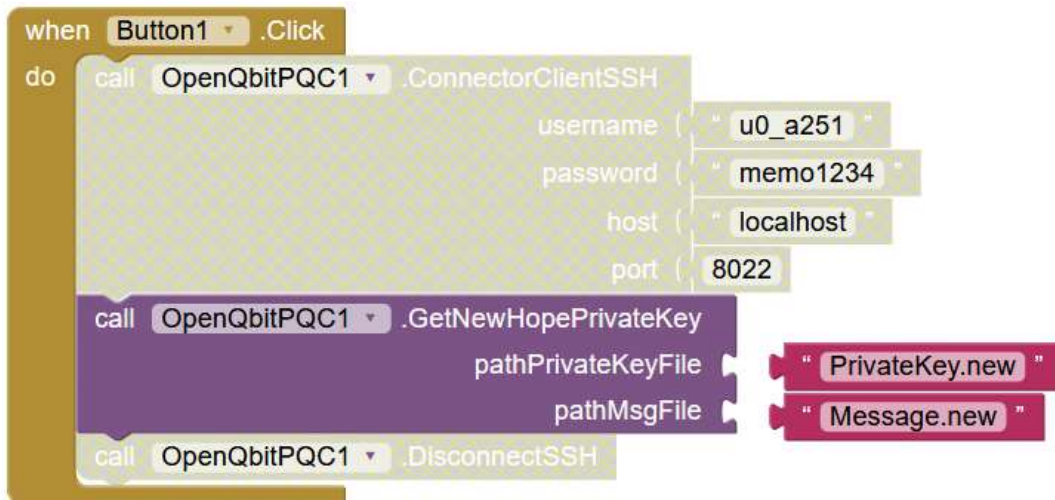
Parámetros de salida: Se ejecuta el evento (**OutputDataNtru**) con el parametro de salida **TextDecNtru** que es el texto original descifrado que contiene el archivo de texto.



En nuestro caso se introdujo el archivo hola.txt el resultado seria un archivo con el nombre: **encrypted.hola.txt**

Descripción: Cifra archivos de texto pequeños estos deben tener **máximo 100 caracteres**.

Bloque para generar llave privada con algoritmo NEWHOPE (**GetNewHopePrivateKey**)



Parámetros de entrada: **pathNewHopePrivateKey<String>**, **pathMsgFile<String>**

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Se ejecuta el evento (**OutputDataNewHopePrivateKeyandMsg**) con el parametro de salida **cipherNewText** que es el contenido del archivo dando en el parámetro de entrada **pathNewHopePrivateKey** concatenado con el contenido del parámetro de entrada **pathMsgFile**.

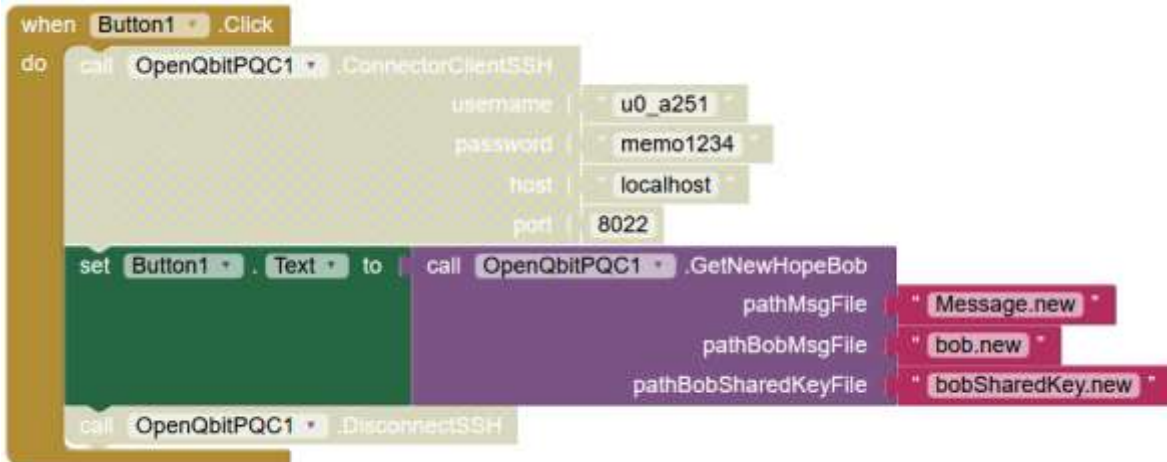


La salida tiene el siguiente formato:

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

Descripción: Genera una llave privada y un mensaje y las guarda en dos archivos separados según la ruta "path" que se dio en los parámetros de entrada, que servirán para posteriormente usarlo entre dos usuarios y verificar las llaves que se compartirán.

Bloque para generar llave de usuario generico "Bob" (GetNewHopeBob)



Parámetros de entrada: **pathMsgFile<String>**, **pathBobMsgFile<String>**, **pathBobSharedKeyFile<String>**

Dependencia obligatoria: Bloque (ConnectorClientSSH), Bloque (DisconnectSSH), Bloque (GetNewHopePrivateKey).

Parámetros de salida: Se crean los archivos dados en los parámetros **pathBobMsgFile** y **pathBobSharedKeyFile** y se entrega una salida del archivo del nombre dado en el parámetro de entrada **pathBobSharedKeyFile**.

La salida tiene el contenido del archivo del parámetro **pathBobSharedKeyFile** con el siguiente formato:

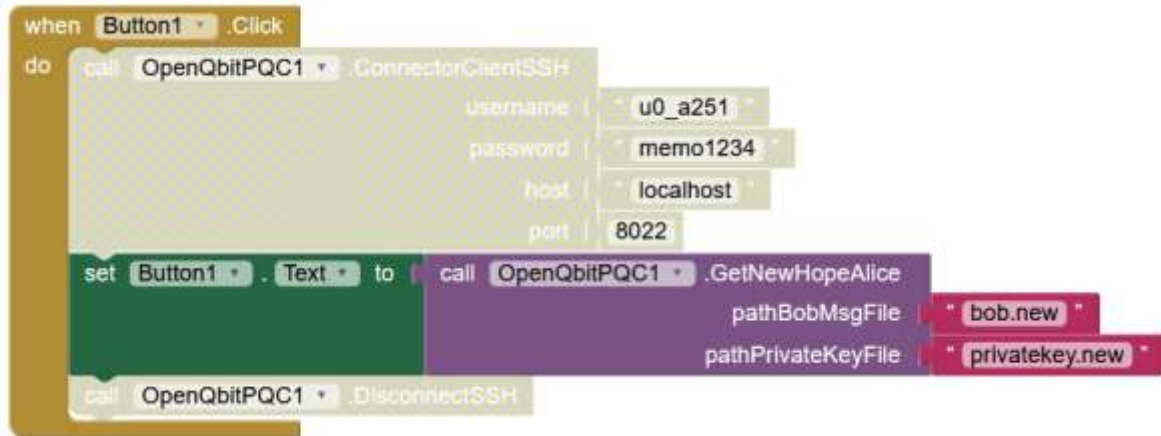
"-----SHARED KEY BOB-----", contenido del archivo pathBobSharedKeyFile

Ejemplo:

```
-----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]
```

Descripción: Bloque para generar llave para ser compartida entre usuarios.

Bloque para generar llave de usuario generico "Alice" (**GetNewHopeAlice**)



Parámetros de entrada: **pathBobMsgFile**<String>, **pathPrivateKeyFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**), Bloque (**GetNewHopeBob**).

Parámetros de salida: Se crean la llave para compartir del segundo usuario en este caso usamos un nombre genérico "Alice".

La salida tiene el contenido de la llave a compartir con el siguiente formato:

```
"-----SHARED KEY ALICE-----", aliceSharedKey
```

Ejemplo:

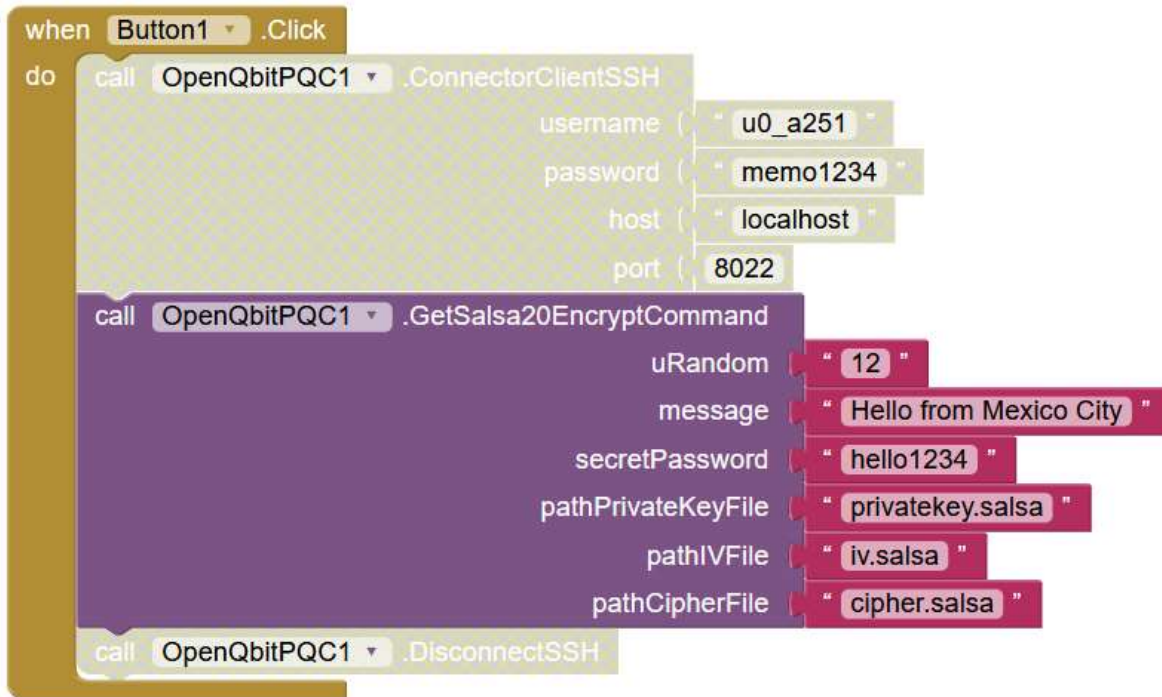
```
-----SHARED KEY ALICE----- [92, 122, 75, 33, 239, 164, 84, 241, 245, 204,
106, 197, 142, 230, 28, 189, 54, 112, 190, 124, 176, 66, 129, 69, 108,
66, 110, 42, 115, 70, 17, 107]
```

Descripción: Bloque para generar llave para ser compartida entre usuarios.

CONFIRMACION DE NEWHOPE: Para confirmar que el intercambio de llaves es correcto, los resultados de los boques (**GetNewHopeBob**) y (**GetNewHopeAlice**) deben ser iguales.

El algoritmo NEWHOPE solo es un **intercambio de llaves** (listas de números) que ayuda a validar información enviada y puede por ejemplo ser usada como password de un método de cifrado como AES.

Boque para cifrar mensaje con algoritmo Salsa20 (**GetSalsa20EncryptCommand**)



Parámetros de entrada: **uRandom**<String>, **message**<String>, **secretPassword**<String>, **pathPrivateKeyFile**<String>, **pathIVFile**<String>, **pathCipherFile**<String>

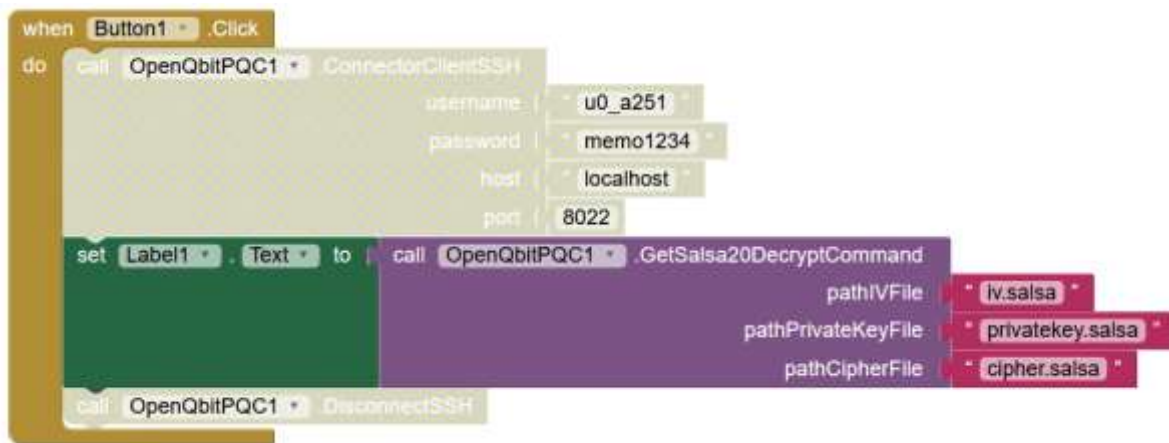
Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Se ejecuta el evento (**OutputDataSalsa20**) con el parametro de salida cipherTextS que es el mensaje cifrado.



Descripción: Bloque para cifrar un mensaje con algoritmo Salsa20.

Boque para descifrar mensaje con algoritmo Salsa20 (**GetSalsa20DecryptCommand**)



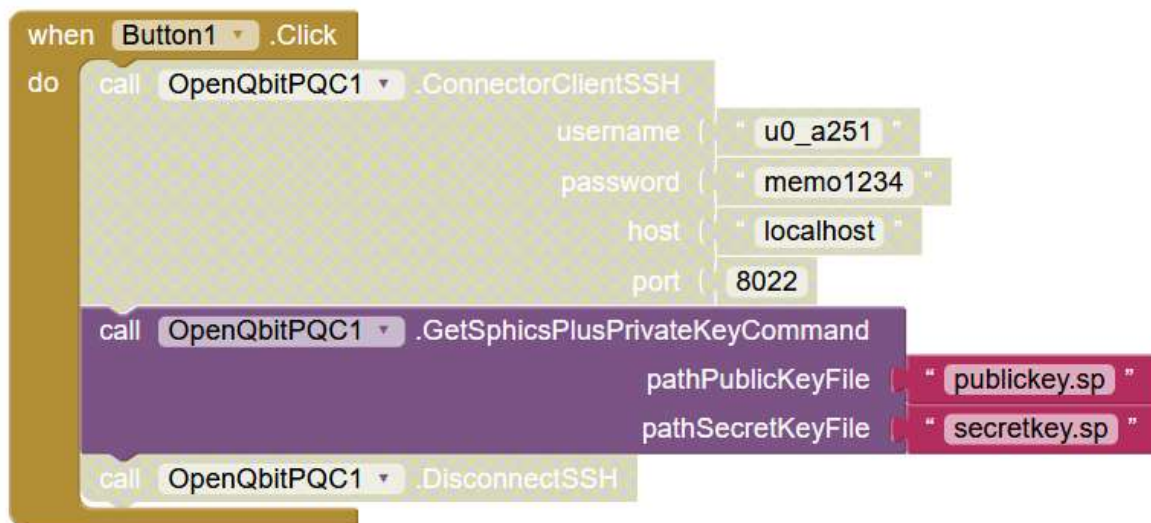
Parámetros de entrada: **pathIVFile**<String>, **pathPrivateKeyFile**<String>, **pathCipherFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Entrega en mensaje descifrado original.

Descripción: Bloque para descifrar un mensaje con algoritmo Salsa20.

Bloque para generar llave privada con el algoritmo Sphics+ (**GetSphicsPlusPrivateKey**)



Parámetros de entrada: **pathPublicKeyFile**<String>, **pathSecretKeyFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Se ejecuta el evento (**OutputDataSphticsPlus**) con el parametro de salida cipherTextSP que entrega la llave publica y llave secreta concatenadas.

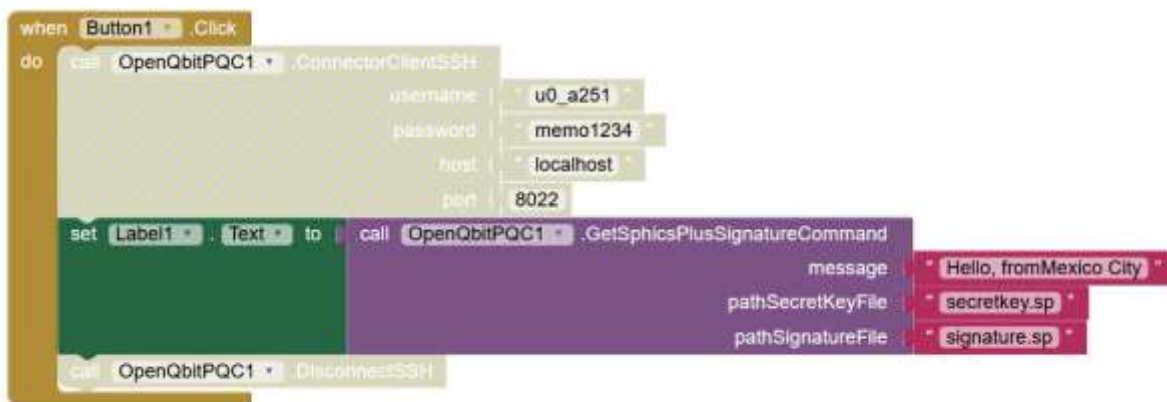


El formato de salida es:

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

Descripción: Genera una llave privada y llave secreta y las guarda en dos archivos separados según la ruta "path" que se dio en los parámetros de entrada.

Bloque para firmar un mensaje con el algoritmo Sphtics+ (**GetSphticsPlusSignature**)



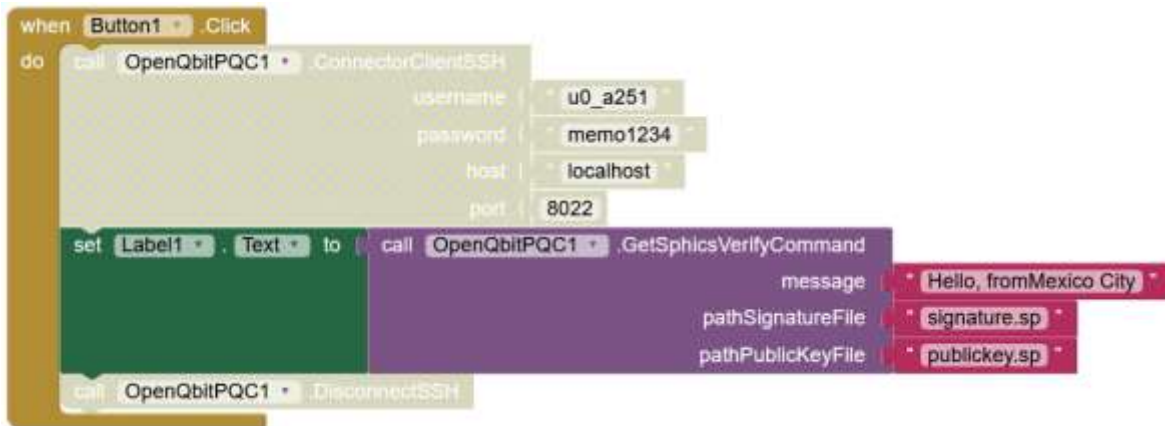
Parámetros de entrada: **message**<String>, **pathSecretKeyFile**<String>, **pathSignatureFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**), Bloque (**GetSphticsPlusPrivateKey**).

Parámetros de salida: crea el archivo con la firma del mensaje que se dio en el parámetro message.

Descripción: Crea una firma que identifica el mensaje proporcionado.

Bloque para verificar firma de mensaje (**GetSphicsVerifyCommand**)



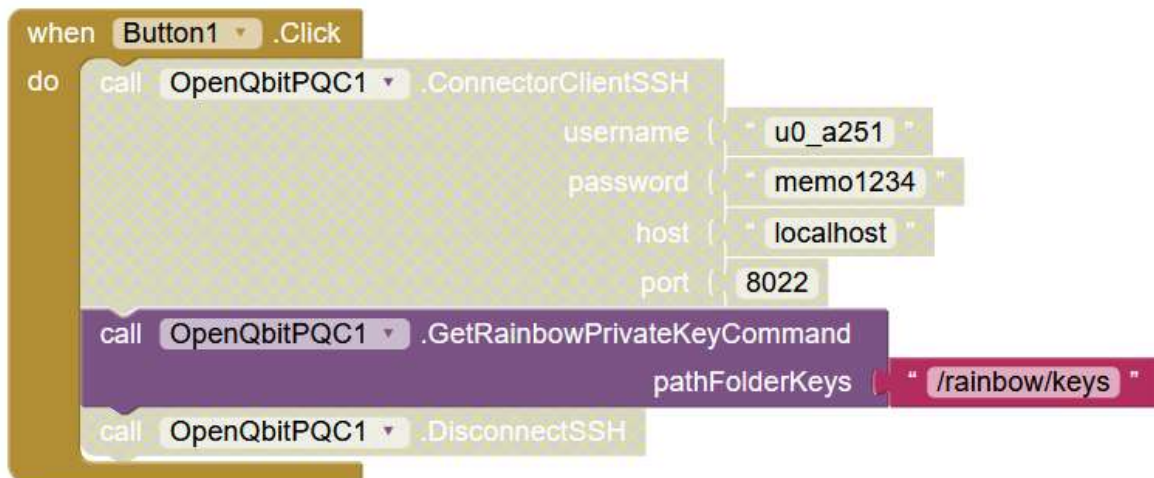
Parámetros de entrada: **message**<String>, **pathSignatureKeyFile**<String>, **pathPublicFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**), Bloque (**GetSphicsPlusPrivateKey**), Bloque (**GetSphicsPlusSignature**).

Parámetros de salida: Verifica firma del mensaje que se dio en el parámetro message.

Descripción: Verifica una firma que identifica el mensaje proporcionado.

Bloque para crear llave privada con algoritmo Rainbow (**GetRainbowPrivateKey**)



Parámetros de entrada: **pathPublicKeyFile**<String>, **pathSecretKeyFile**<String>

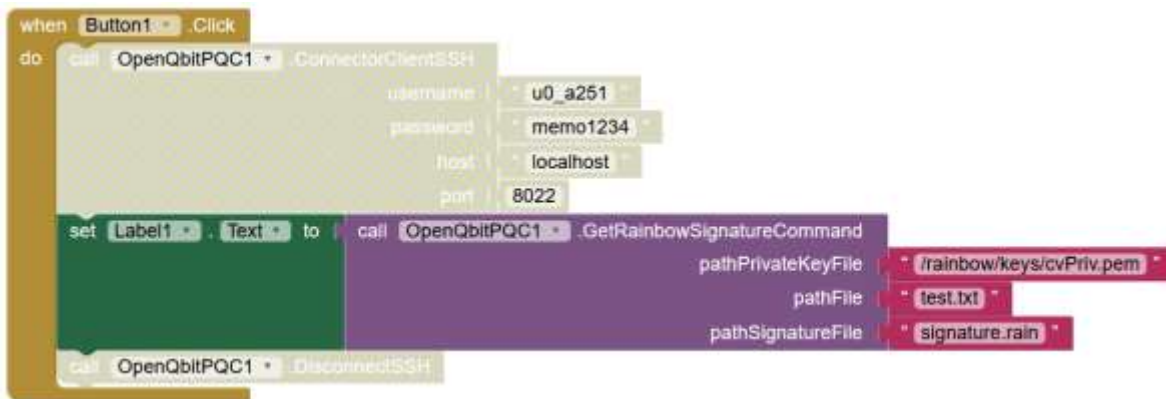
Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**).

Parámetros de salida: Se ejecuta el evento (**OutputDataRainbow**) con el parametro de salida cipherTextR que entrega un listado del directorio en donde se crearon la llave privada y llave publica.



Descripción: Genera una llave privada y llave publica y las guarda en dos archivos separados según la ruta del folder “path” que se dio en los parámetros de entrada.

Bloque para firmar archivo con algoritmo Rainbow(**GetRainbowSignatureCommand**)



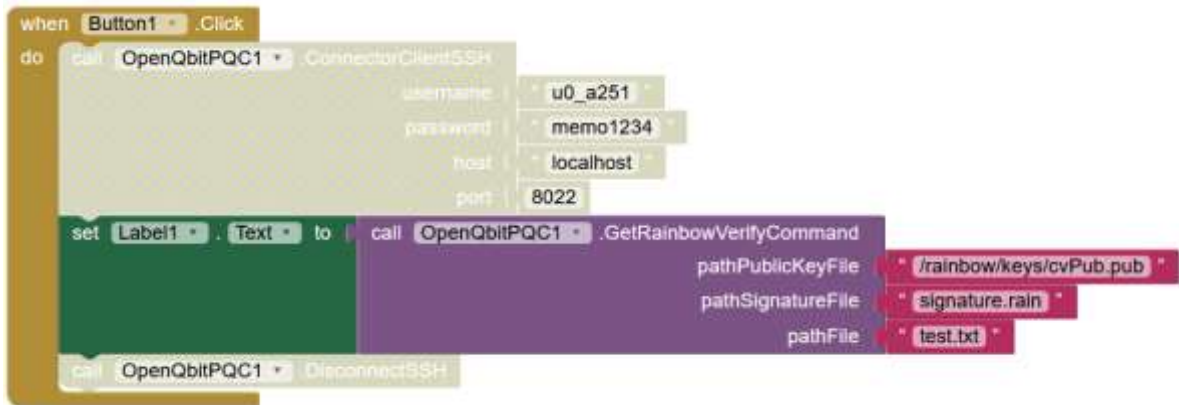
Parámetros de entrada: **pathPrivateKeyFile** <String>, **pathFile**<String>, **pathSignatureFile**<String>

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**), Bloque (**GetRainbowPrivateKeyCommand**).

Parámetros de salida: Crea el archivo con la firma del archivo que se dio en el parámetro **pathSignatureFile**.

Descripción: Verifica una firma que identifica el archivo proporcionado.

Boque para verificar firma de archivo (**GetRainbowVerifyCommand**)



Parámetros de entrada: **pathPublicKeyFile**<String>, **pathSignatureKeyFile**<String>, **pathFile**<String>.

Dependencia obligatoria: Bloque (**ConnectorClientSSH**), Bloque (**DisconnectSSH**), Bloque (**GetSphicsPlusPrivateKey**), Bloque (**GetSphicsPlusSignature**).

Parámetros de salida: Verifica firma del archivo que se dio en el parámetro de entrada **pathFile**.

Descripción: Verifica una firma que identifica el archivo proporcionado.

11. Anexo “Programas Python”.

Los nombres y ubicaciones no deberán ser cambiados ya que el bloque de comando hace referencia a este nombre y ubicación.

Programas para algoritmo **Chacha20Poly1305** que debern ser instalados en el home de usuario termux default **/data/data/com.termux/files/home** del usuario de la terminal de Termux.

Nombre: **chacha20encrypt.py**

```
import os
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)
```

Nombre: **chacha20decrypt.py**

```
import os
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)
```

Los nombres y ubicaciones no deberán ser cambiados ya que el bloque de comando hace referencia a este nombre y ubicación.

Programas para algoritmo **NEWHOPE** que debern ser instalados en el home de usuario termux default `/data/data/com.termux/files/home` del usuario de la terminal de Termux.

Nombre: **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, fp)

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

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

Nombre: **newhope-bob.py**

```
from pynewhope import newhope
import pickle
import sys

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

bobSharedKey, bobMsg = newhope.sharedB(Msg)

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

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

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

Nombre: **newhope-alice.py**

```
from pynewhope import newhope
import pickle
import sys

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

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

AliceSharedKey = newhope.sharedA(bobMsg, PrivKey)

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

Crear en App inventor el comparativo siguiente para confirmar que el intercambio de claves entre Bob y Alice (nombre de usuarios ejemplo) es correcto.

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

Los nombres y ubicaciones no deberán ser cambiados ya que el bloque de comando hace referencia a este nombre y ubicación.

Programas para algoritmo **SALSA20** que debern ser instalados en el home de usuario termux default `/data/data/com.termux/files/home` del usuario de la terminal de Termux.

Nombre: **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

import os
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)
```


Nombre: **salsa20decrypt.py**

```
from salsa20 import XSalsa20_xor

import binascii
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

import os
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())
```

Los nombres y ubicaciones no deberán ser cambiados ya que el bloque de comando hace referencia a este nombre y ubicación.

Programas para algoritmo **SPHINCS+** que debern ser instalados en el home de usuario termux default **/data/data/com.termux/files/home** del usuario de la terminal de Termux.

Nombre: **sphicsplus-privatekey.py**

```
import pyspx.shake256_128f
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

import os
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)
```

Nombre: sphicsplus-signature.py

```
import pyspx.shake256_128f
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

import os
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)
```

Nombre: sphicsplus-verify.py

```
import pyspx.shake256_128f
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

import os
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
```

Los nombres y ubicaciones no deberán ser cambiados ya que el bloque de comando hace referencia a este nombre y ubicación.

Programas para algoritmo **RAINBOW** que debern ser instalados en el home de usuario termux default `/data/data/com.termux/files/home` del usuario de la terminal de Termux.

Nombre: **rainbow-privatekey.py**

```
from cryptovinaigrette import cryptovinaigrette

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

Nombre: **rainbow-signature.py**

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

import os
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)

Nombre: rainbow-verify.py

```
from cryptovinaigrette import cryptovinaigrette
import base64
import hashlib

from base64 import standard_b64encode
from base64 import standard_b64decode

import os
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!")
```

Cifrado de archivos con **CODECRYPT** esta suite tiene una variedad de algoritmos Post-Quantum Computing (PQC). La usaremos directamente en línea de comando a través del servicio de SSH (Secure Shell), con bloque (**OpenQbitConnectSSH**).

Ejemplo de uso de comandos:

```
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

#verify 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
```

Para ver detalle de comandos consultar las siguientes ligas:

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

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

Manual de usuario (Command Line) CODECRYPT

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

12. Anexo “Computación Cuántica con OpenQbit”.

¿Cómo funciona la computación cuántica?

La transformación digital está generando cambios en el mundo más rápido que nunca. ¿Creerías que la era digital está por acabarse? Ya se ha señalado la **alfabetización digital** como un área donde el conocimiento abierto y las oportunidades accesibles para aprender sobre la tecnología son urgentes para abordar las brechas en el desarrollo social y económico. Aprender de los conceptos claves de la era digital se volverá aún más crítico ante la inminente llegada de otra nueva ola tecnológica capaz de transformar los modelos existentes con una velocidad y potencia asombrosa: las **tecnologías cuánticas**.

En este artículo, comparamos los conceptos básicos de la computación tradicional y la computación cuántica; y también comenzamos a explorar su aplicación en otras áreas relacionadas.

¿Qué son las tecnologías cuánticas?

A lo largo de la historia, el ser humano ha ido desarrollando tecnología a medida que ha ido entendiendo el funcionamiento de la naturaleza a través de la ciencia. Entre los años 1900 y 1930, el estudio de algunos fenómenos físicos que aún no estaban bien entendidos dio lugar a una nueva teoría física, la **Mecánica Cuántica**. Esta teoría describe y explica el funcionamiento del mundo microscópico, hábitat natural de moléculas, átomos o electrones. Gracias a ella no solo se ha conseguido explicar esos fenómenos, sino que ha sido posible entender que la realidad subatómica funciona de forma completamente contra intuitiva, casi mágica, y que en el mundo microscópico tienen lugar sucesos que no ocurren en el mundo macroscópico.

Entre estas **propiedades cuánticas** se incluyen la superposición cuántica, el entrelazamiento cuántico y el teletransporte cuántico.

- La **superposición cuántica** describe cómo una partícula puede estar en diferentes estados a la vez.
- El **entrelazamiento cuántico** describe cómo dos partículas tan separadas como se desee pueden estar correlacionadas de forma que, al interactuar con una, la otra se entera.
- El **teletransporte cuántico** utiliza el entrelazamiento cuántico para enviar información de un lugar a otro del espacio sin necesidad de viajar a través de él.

Las tecnologías cuánticas son basadas en estas propiedades cuánticas de la naturaleza subatómica.

En este caso, hoy en día el entendimiento del mundo microscópico a través de la Mecánica Cuántica nos permite inventar y diseñar tecnologías capaces de mejorar la vida de las personas. Hay muchas y muy diferentes tecnologías que utilizan fenómenos cuánticos y, algunas de ellas, como el láser o las imágenes por resonancia magnética (IRM), llevan ya entre nosotros más de medio siglo. Sin embargo, actualmente estamos presenciando una revolución tecnológica en áreas como la computación cuántica, la información cuántica, la simulación cuántica, la óptica cuántica, la metrología cuántica, los relojes cuánticos o los sensores cuánticos.

¿Qué es la computación cuántica? Primero, hay que entender la computación clásica.




FIGURA 1.
Ejemplos de caracteres en lenguaje binario.

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

Para entender cómo funcionan los computadores cuánticos es conveniente explicar primero cómo funcionan los computadores que utilizamos a diario, a los que nos referiremos en este documento como computadores digitales o clásicos. Estos, al igual que el resto de los dispositivos electrónicos como tabletas o teléfonos móviles, utilizan bits como unidades fundamentales de memoria. Esto significa que los programas y aplicaciones están codificados en bits, es decir, en lenguaje binario de ceros y unos. Cada vez que interactuamos con cualquiera de estos dispositivos, por ejemplo, pulsando una tecla del teclado, se crean, destruyen y/o modifican cadenas de ceros y unos dentro de la computadora.

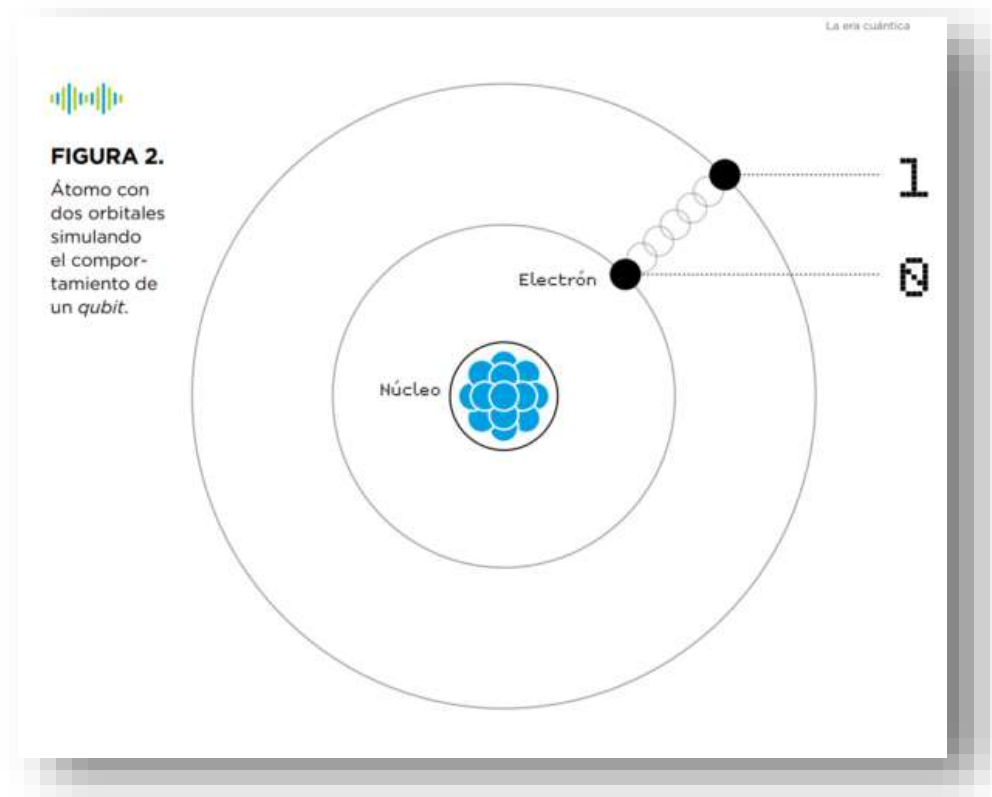
La pregunta interesante es, ¿qué son físicamente estos ceros y unos dentro de la computadora? Los estados cero y uno de los bits se corresponden con corriente eléctrica que circula, o no, a través de unas piezas microscópicas denominadas transistores, que actúan como interruptores. Cuando no circula corriente, el transistor está “apagado” y se corresponde con un bit 0, y cuando circula está “encendido” y se corresponde con un bit 1.

De forma más simplificada, es como si los bits 0 y 1 se correspondiesen con huecos, de manera que un hueco vacío es un bit 0 y un hueco ocupado por un electrón es un bit 1. Es por este motivo que estos dispositivos se llaman electrónicos. A modo de ejemplo, en la figura 1 se muestra la escritura en lenguaje binario de algunos caracteres. Ahora que

tenemos una idea de cómo funcionan los computadores actuales, tratemos de entender cómo funcionan los cuánticos.

De los bits a qubits

La unidad fundamental de información en computación cuántica es el quantum bit o qubit. Los qubits son, por definición, sistemas cuánticos de dos niveles -ahora veremos ejemplos- que al igual que los bits pueden estar en el nivel bajo, que se corresponde con un estado de baja excitación o energía definido como 0, o en el nivel alto, que se corresponde con un estado de mayor excitación o definido como 1. Sin embargo, y aquí radica la diferencia fundamental con la computación clásica, los qubits también pueden estar en cualquiera de los infinitos estados intermedios entre el 0 y el 1, como por ejemplo un estado que sea mitad 0 y mitad 1, o tres cuartos de 0 y un cuarto de 1. Este fenómeno se conoce como superposición cuántica y es natural en sistemas cuánticos.



Algoritmos cuánticos, computación exponencialmente más poderosa y eficiente

El propósito de los computadores cuánticos es aprovechar estas propiedades cuánticas de los *qubits*, como sistemas cuánticos que son, para poder correr algoritmos cuánticos que utilizan la superposición y el entrelazamiento para ofrecer una capacidad de procesamiento mucho mayor que los clásicos. Es importante indicar que el verdadero cambio de paradigma no consiste en hacer lo mismo que hacen las computadoras digitales o clásicas -las actuales-, pero más rápido, como de forma errónea se puede leer en muchos artículos, sino que los

[OpenQubit.com](https://openqubit.com)

algoritmos cuánticos permiten realizar ciertas operaciones de una manera totalmente diferente que en muchos casos resulta ser más eficiente -es decir, en mucho menos tiempo o utilizando muchos menos recursos computacionales-.

Veamos un ejemplo concreto de lo que esto implica. Imaginemos que estamos en Bogotá y queremos saber cuál es la mejor ruta para llegar a Lima de entre un millón de opciones para llegar ($N=1.000.000$). De cara a poder utilizar computadoras para encontrar el camino óptimo necesitamos digitalizar 1.000.000 opciones, lo que implica traducirlas a lenguaje de bits para el computador clásico y a *qubits* para el computador cuántico. Mientras que una computadora clásica necesitaría ir uno por uno analizando todos los caminos hasta encontrar el deseado, una computadora cuántica se aprovecha del proceso conocido como paralelismo cuántico que le permite considerar todos los caminos a la vez. Esto implica que, si bien la computadora clásica necesita del orden de $N/2$ pasos o iteraciones, es decir, 500.000 intentos, la computadora cuántica encontrará la ruta óptima tras solo \sqrt{N} operaciones sobre el registro, es decir, 1.000 intentos.

En el caso anterior la ventaja es cuadrática, pero en otros casos es incluso exponencial, lo que significa que con n *qubits* podemos obtener una capacidad computacional equivalente a 2^n bits. Para ejemplificar esto, es frecuente contar que con unos 270 qubits se podrían tener más estados base en un computador cuántico -más cadenas de caracteres diferentes y simultáneas- que el número de átomos en el universo, que se estima en torno a 10^{80} . Otro ejemplo, es que se estima que con un computador cuántico de entre 2000 y 2500 *qubits* se podría romper prácticamente toda la criptografía utilizada hoy en día (la conocida como criptografía de clave pública).

¿Por qué es importante saber sobre la tecnología cuántica?

Estamos en un momento de transformación digital en el que distintas tecnologías emergentes como blockchain, inteligencia artificial, drones, Internet de las cosas, realidad virtual, 5G, impresoras 3D, robots o vehículos autónomos tienen cada vez más presencia en múltiples ámbitos y sectores. Estas tecnologías, llamadas a mejorar la calidad de vida del ser humano acelerando el desarrollo y generando impacto social, avanzan hoy en día de manera paralela. Solo en contadas ocasiones vemos compañías desarrollando productos que exploten combinaciones de dos o más de estas tecnologías, como blockchain y IoT o drones e inteligencia artificial. Si bien están destinadas a converger generando así un impacto exponencialmente mayor, la etapa inicial de desarrollo en que se encuentran y la escasez de desarrolladores y personas con perfiles técnicos hacen que las convergencias sean aún una tarea pendiente.

De las tecnologías cuánticas, por su potencial disruptivo, se espera que no solo converjan con todas estas nuevas tecnologías, sino que tengan una influencia transversal en prácticamente la totalidad de ellas. La computación cuántica amenazará la autenticación, intercambio y almacenamiento seguro de datos, teniendo un impacto mayor en aquellas

tecnologías en las que la criptografía tiene un rol más relevante, como ciberseguridad o blockchain, y un impacto negativo menor pero también a considerar en tecnologías como 5G, IoT o drones.

¿Quieres practicar la computación cuántica?

En la red hay ya disponibles decenas de simuladores de computadores cuánticos con diferentes lenguajes de programación ya existentes como C, C++, Java, Matlab, Máxima, Python o Octave. También, lenguajes nuevos como Q#, lanzado por Microsoft. Se puede explorar y jugar con una máquina cuántica virtual a través de plataformas como la de IBM y la de Rigetti.

Mini QRNG es creado por OpenQbit.com compañía que se enfoca en desarrollar tecnología basada en computación cuántica para diferentes tipos de sectores tanto privado como público.

Por qué Mini QRNG es diferente a los demás QRNG, simplemente porque el sistema fue creado para ser modular y se puede armar fácilmente en casa con un costo bastante económico.

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

13. Licenciamiento y uso de 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 y App Inventor Blockly

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

Extensiones externas:

JSOINTOOLS

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

Licenciamiento versiones opensource y comercial de sistema Mini QRNG consultar la página oficial <http://www.openqbit.com>

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

Mini PQC (Post-Quantum Cryptography) es dominio público.

Todo el código y la documentación en Mini PQC ha sido dedicado al dominio público por los autores. Todos los autores de códigos y representantes de las empresas para las que trabajan han firmado declaraciones juradas dedicando sus contribuciones al dominio público y los originales de esas declaraciones juradas se almacenan en una caja fuerte en las oficinas principales de OpenQbit México. Cualquier persona es libre de publicar, usar o distribuir las extensiones de Mini PQC (OpenQbit) original, ya sea en forma de código fuente o como binario compilado, para cualquier propósito, comercial o no comercial, y por cualquier medio.

El párrafo anterior se aplica al código y la documentación entregables en Mini PQC aquellas partes de la biblioteca de Mini PQC que realmente agrupa y envía con una aplicación más grande. Algunos scripts utilizados como parte del proceso de compilación (por ejemplo, los

scripts de "configuración" generados por autoconf) pueden estar incluidos en otras licencias de código abierto. Sin embargo, nada de estos scripts de compilación llega a la biblioteca entregable final de Mini PQC, por lo que las licencias asociadas con esos scripts no deberían ser un factor en la evaluación de sus derechos para copiar y usar la biblioteca Mini PQC.

Todo el código entregable en Mini PQC ha sido escrito desde cero. No se ha tomado ningún código de otros proyectos o de Internet abierto. Cada línea de código puede rastrearse hasta su autor original, y todos esos autores tienen dedicaciones de dominio público en el archivo. Por lo tanto, la base de código Mini PQC es limpia y no está contaminada con código con licencia de otros proyectos de código abierto, no contribución abierta

Mini QRNG es de código abierto, lo que significa que puede hacer tantas copias como desee y hacer lo que quiera con esas copias, sin limitación. Pero Mini PQC no es de contribución abierta. Para mantener Mini PQC en el dominio público y garantizar que el código no se contamine con contenido patentado o con licencia, el proyecto no acepta parches de personas desconocidas. Todo el código en Mini PQC es original, ya que ha sido escrito específicamente para su uso por Mini PQC. No se ha copiado ningún código de fuentes desconocidas en Internet.

Mini PQC es de dominio público y no requiere una licencia. Aun así, algunas organizaciones quieren pruebas legales de su derecho a usar Mini PQC. Las circunstancias donde esto ocurre incluyen lo siguiente:

- Su empresa desea indemnización por reclamos de infracción de derechos de autor.
- Está utilizando Mini PQC en una jurisdicción que no reconoce el dominio público.
- Está utilizando de Mini PQC en una jurisdicción que no reconoce el derecho de un autor a dedicar su trabajo al dominio público.
- Desea tener un documento legal tangible como evidencia de que tiene el derecho legal de usar y distribuir de Mini PQC.
- Su departamento legal le dice que debe comprar una licencia.

Si alguna de las circunstancias anteriores se aplica a usted, OpenQbit, la compañía que emplea a todos los desarrolladores de Mini PQC, le venderá una Garantía de título para Mini PQC. Una garantía de título es un documento legal que afirma que los autores reclamados de Mini PQC son los verdaderos autores, y que los autores tienen el derecho legal de dedicar el Mini PQC al dominio público, y que OpenQbit se defenderá enérgicamente contra los reclamos de licenciamiento. Todos los ingresos de la venta de las garantías de título de Mini PQC se utilizan para financiar la mejora continua y el soporte de Mini PQC.

Código contribuido

Para mantener Mini PQC completamente libre y libre de derechos de autor, el proyecto no acepta parches. Si desea hacer un cambio sugerido e incluir un parche como prueba de concepto, sería genial. Sin embargo, no se ofenda si reescribimos su parche desde cero. El tipo de licenciamiento no-comercial u opensource quien lo use en esta modalidad y alguna similar sin compra de soporte ya sea uso individual o corporativo no importando el tamaño de empresa se rigira por las siguientes premisas legales.

Renuncia de garantía. A menos que lo exija la ley aplicable o acordado por escrito, el Licenciante proporciona el Trabajo (y cada El Colaborador proporciona sus Contribuciones) "TAL CUAL", **SIN GARANTÍAS O CONDICIONES DE NINGÚN TIPO**, ya sea expreso o implícito, incluidas, entre otras, cualquier garantía o condición de TÍTULO, NO INFRACCIÓN, COMERCIALIZACIÓN O APTITUD PARA UN PROPÓSITO PARTICULAR. Usted es el único responsable de determinar el correcto uso o redistribuir el Trabajo y asumir cualquier riesgo asociado con su ejercicio de permisos bajo esta Licencia.

Cualquier pérdida económica o de algún tipo por el uso de este software el afectado no tendrá opción de pago de ningún tipo. Todo litigio legal las partes se someterán a tribunales únicamente en la jurisdicción de la Ciudad de México, país México.

Para soporte, uso y licenciamiento comercial se tendrá que realizar un acuerdo o contrato establecido entre OpenQbit o su corporativo y la parte interesada.

Los términos y condiciones de distribución comercialización pueden cambiar sin previo aviso, dirigirse al portal oficial de www.openqbit.com para ver cualquier modificación de cláusulas de soporte y licenciamiento no-comercial y comercial.

Cualquier persona, usuario, entidad privada, publica de cualquier índole legal o de cualquier parte del mundo quien simplemente use el software acepta sin condicionantes las clausulas establecidas en este documento y las que se puedan modificar en cualquier momento en el portal de www.openqbit.com sin previo aviso pudiendo ser aplicadas a discreción de OpenQbit en uso no-comercial o comercial.

Cualquier duda e información de Mini PQC dirigirse a la comunidad de App Inventor o a las comunidades de diversos sistemas Blockly como son: AppBuilder, Trunkable, etc. y/o al correo opensource@openqbit.com por la demanda de preguntas puede tardar la respuesta de 3 a 5 días hábiles.

Soporte con uso comercial.

support@openqbit.com

Ventas uso comercial.

sales@openqbit.com

Información legal y preguntas o dudas de licenciamiento.

legal@openqbit.com

