Instalación y Configuración de una Red Privada para la Mini-Blockchain *Cryptonite*

Sebastián Suarez[1], Miguel A. Astor[1], Antonio Russoniello[1], Ana Morales Bezeira[1], Wilmer Pereira[2]
sebastian.suarez@ciens.ucv.ve, miguel.astor@ciens.ucv.ve, antonio.russoniello@ciens.ucv.ve, ana.morales@ciens.ucv.ve, wpereira@ucab.edu.ve

[1] Centro CICORE, Escuela de Computación, Universidad Central de Venezuela, Caracas, Venezuela
[2] Departamento de Computación, Instituto Tecnológico Autónomo de México, Ciudad de México, México

Resumen: Las criptomonedas y otros sistemas basados en la tecnología Blockchain sufren de un problema de escalabilidad con respecto al uso del espacio de almacenamiento, el cual tiene como consecuencia principal una tendencia a la centralización que contradice el objetivo de descentralización que funge como principal motivación de esta tecnología. Una posible solución a este problema es el esquema de Mini-Blockchain, representado por su implementación de referencia, la criptomoneda *Cryptonite*. Sin embargo, este esquema no tiene soporte para la programación del comportamiento de la verificación de transacciones como en *Bitcoin*, o la ejecución de contratos inteligentes como en *Ethereum*, lo que ha derivado en un proyecto para incorporar este soporte al sistema *Cryptonite*. Como complemento a dicho proyecto, en este trabajo se presenta un manual de configuración del sistema de Mini-Blockchain *Cryptonite* para la instalación de redes privadas desconectadas de la red principal de dicha criptomoneda, funcionalidad que no se encuentra implementada directamente en el proyecto *Cryptonite*.

Palabras Clave: Criptomonedas; Blockchain; Escalabilidad; Mini-Blockchain; Cryptonite; Red Privada.

I. Introducción

Desde sus orígenes en la especificación del sistema *Bitcoin* por Satoshi Nakamoto en [1], las criptomonedas se han presentado como una innovadora alternativa descentralizada al sistema financiero tradicional. Así mismo, la tecnología *Blockchain* central a estos sistemas ha derivado en una gran producción de investigación y desarrollo que buscan aplicar dicha tecnología a procesos de negocios que requieren de una administración centralizada de registros de transacciones [2]–[5].

Sin embargo, la *Blockchain* sufre de varios problemas de escalabilidad como resultado de las consideraciones tomadas por Nakamoto al momento de diseñarla [2]–[4]. En particular, al ser una estructura de datos de solo-añadidura, esta crece constantemente a lo largo del tiempo lo que crea problemas para su almacenamiento y dificulta el proceso de compartirla entre los participantes de una red basada en *Blockchain*. Este problema también implica dificultades para la sincronización de nuevos pares en la red, provocando que esta tienda a la centralización en pocos nodos completos capaces de contribuir a la misma, mientras que la mayoría de los usuarios dependen de nodos ligeros sujetos al consenso de los nodos completos [6]. Este problema de crecimiento no acotado se conoce como *Blockchain bloat* [7].

Una solución al *Blockchain bloat* conocida como esquema de *Mini-Blockchain* se define en [8]. El esquema de *Mini-Blockchain* plantea una separación de las funcionalidades de la *Blockchain* en tres estructuras de datos de las cuales dos pueden descartar información cuando esta deja de ser necesaria; mientras que la estructura restante, que sí crece indefinidamente como la *Blockchain* normal, está diseñada para contener solamente el mínimo de información necesaria para cumplir sus funciones.

Todo sistema basado en este esquema es incapáz de soportar procesos relacionados a la *Blockchain* que dependan de infomación aparte de los estados de cuenta, como un efecto secundario de la capacidad de descartar información de la *Mini-Blockchain*. De esta manera la *Mini-Blockchain* es incapaz de soportar transacciones ejecutables, como es el caso de *Bitcoin* con su lenguaje Script; o la ejecución de contratos inteligentes como plantea la criptomoneda *Ethereum* [9], [10].

En base a esto, en 2018 Sanguña et al. [11] describen una propuesta de incorporación de soporte para la ejecución de contratos inteligentes al esquema de *Mini-Blockchain*, usando la implementación *Cryptonite* [12] como base para el desarrollo de una prueba de concepto de dicha propuesta. Como parte del desarrollo de esta propuesta se requiere de la capacidad de poder crear un entorno de pruebas basado en *Cryptonite*, para lo cual se presenta en este trabajo un manual de adaptación del código fuente del sistema para la creación de redes privadas, funcionalidad que no está soportada por la versión actual de *Cryptonite*.

El resto de este trabajo se estructura de la siguiente manera. En la Sección II se presenta un breve marco teórico donde se describe el origen de las criptomonedas y la tecnología *Blockchain* que las sustenta, y que da pie al esquema de *Mini-Blockchain* como solución a sus problemas de escalabilidad en disco. La Sección III describe el procedimiento de modificación y compilación del código fuente del sistema *Cryptonite* para la creación de redes privadas para la criptomoneda basada en *Mini-Blockchain*. Finalmente, la Sección IV contiene las

conclusiones y propuestas de trabajos futuros.

II. Blockchain y *Mini-Blockchain*

En esta Sección se describirán brevemente los hitos que llevaron al desarrollo de la tecnología *Blockchain* y el posterior desarrollo de la *Mini-Blockchain* como esquema de solución a ciertos problemas de escalabilidad presentados por la *Blockchain*.

II-A. Bitcoin y los Orígenes de la Blockchain

Las criptomonedas y la tecnología de *Blockchain* tienen su origen en la publicación de la especificación del sistema Bitcoin por Satoshi Nakamoto en el año 2008 [1]. Este artículo describe un sistema financiero distribuido llamado *Bitcoin*, el cual se basa en firmas digitales y marcas de tiempo encadenadas en una estructura de datos de solo-añadidura actualmente conocida como *Blockchain*. A diferencia de otros sistemas financieros electrónicos, *Bitcoin* fue diseñado para permitir el intercambio de monedas digitales entre usuarios evitando el llamado problema del *double-spending* (gasto doble) sin necesidad de una tercera parte confiable que verifique la validez de la transacciones.

Sin embargo, *Bitcoin* no es ni mucho menos el primer intento de desarrollar un sistema financiero electrónico con dichas características, y sus innovaciones tecnológicas tampoco fueron desarrolladas en un vacío. Antecedentes directos en el área de sistemas financieros electrónicos descentralizados existen en los proyectos *b-money* de Dai [13], y los trabajos de Szabo en el área de intercambio de activos digitales [14] que culminaron en su especificación del sistema *Bit Gold* [15].

A partir de estos proyectos, Nakamoto propuso entonces un sistema financiero electrónico completamente distribuido, mediante el cual los usuarios del sistema pueden generar valor verificando las transacciones que realizan los otros usuarios mediante la aplicación de algoritmos de prueba de trabajo a bloques (conjuntos) de tamaño fijo, compuestos de transacciones. Estos bloques son enlazados criptográficamente en una estructura de datos que actúa como el libro mayor contable del sistema. Esta estructura de datos es conocida hoy en día como *Blockchain*[1] y cumple tres funciones cruciales para el funcionamiento de las criptomonedas [6]:

- Coordinar el procesamiento de transacciones por parte de los nodos participantes de la red.
- Asegurar su propia integridad mediante el mecanismo de prueba de trabajo.
- Almacenar y administrar el balance de las cuentas de los usuarios de la red.

La idea de encadenar bloques de datos utilizando técnicas criptográficas puede encontrarse en la literatura desde hace muchas décadas, siendo el trabajo de Ehrsam et al. [16] posiblemente la publicación más temprana en utilizar el concepto. Trabajos posteriores como el ya mencionado *Bit Gold* [15] desarrollaron la idea.

El gran acierto de Nakamoto consistió en la idea de asociar dos elementos adicionales a esta cadena de bloques [1]: una política completamente descentralizada para alcanzar el consenso sobre el verdadero historial de transacciones, y un algoritmo de verificación de transacciones con incentivos para garantizar el buen comportamiento de los participantes del sistema sin necesidad de un intermediario centralizado confiable. El algoritmo de verificación de transacciones propuesto en [1] se basa directamente en los algoritmos PoW (*Proof-of-Work*, Prueba-de-Trabajo) utilizados por el sistema Hashcash de Back [17], [18], que a su vez se inspiran en el trabajo de Dwork [19].

II-B. Escalabilidad de la Blockchain y la Mini-Blockchain

Una de las características más resaltantes, y a la vez un inconveniente de la *Blockchain* como estructura de datos es que esta crece indefinidamente a lo largo del tiempo; por diseño, no existe un mecanismo para eliminar datos de una *Blockchain*. Este crecimiento no acotado es conocido como *Blockchain bloat* [7].

Una consecuencia del problema del *Blockchain bloat* dentro de cualquier red basada en *Blockchain* radica en el incremento de la centralización de la red. A medida que la *Blockchain* requiere de mayor cantidad de espacio de almacenamiento, se hace cada vez menos factible el que los usuarios del sistema sean capaces de almacenar la estructura de datos completa, concentrándose todo el mecanismo de consenso en un pequeño grupo de "mineros" que si poseen la capacidad de almacenarla [6]. De igual forma, a medida que incrementa el tamaño de la *Blockchain*, los nodos participantes inevitablemente tardan mayor cantidad de tiempo en sincronizarse con la red al momento de unirse al sistema, siendo preferible el uso de mecanismos como SPV (*Simplified Payment Verification*, Verificación de Pagos Simplificada) [3].

Un estudio sistemático de la bibliografía revela que muy poco esfuerzo de investigación ha sido dedicado a buscar soluciones a este problema [2]–[4]. En particular se identifican dos posibles soluciones: el proyecto *Ultimate Blockchain Compression* (Compresión Máxima de la *Blockchain*) [20] el cual fue abandonado en el año 2014; y el esquema de *Mini-Blockchain* [8].

En [6], con revisiones posteriores en [8], [21], J. D. Bruce introdujo el esquema de *Mini-Blockchain* como una propuesta de mecanismo para reducir el crecimiento no acotado en cuanto a uso de espacio de almacenamiento de la *Blockchain* de *Bitcoin*. Este esquema se fundamenta en la observación por parte de Bruce de que, dentro de una red como *Bitcoin*, la *Blockchain* sirve fundamentalmente para cumplir los tres objetivos listados en la Sección II-A. En particular, Bruce observa como no es necesario mantener indefinidamente la totalidad del historial de transacciones en la *Blockchain* cuando las transacciones se interpretan como operaciones simples sobre los estados de cuenta de los usuarios del sistema.

En lineas generales, el esquema de mini-blockchain consiste en separar las tres funcionalidades mencionadas en estructuras

[1]Es de resaltar que Nakamoto no introdujo el término *Blockchain* para referirse a la estructura de datos, llamándola sencillamente "*blocks in a chain*" (bloques en una cadena). El sustantivo *Blockchain* sería acuñado en años posteriores.

de datos distintas pero relacionadas, en lugar de utilizar una única estructura de datos [8]. Estas tres estructuras de datos son conocidas como el *account tree* (árbol de cuentas), la mini-blockchain y la *proof chain* (cadena de pruebas).
El proyecto *Cryptonite* [12] representa la primera implementación del esquema de mini-blockchain según se define en [8]. *Cryptonite* se basa en el código fuente de *Bitcoin*, el cual ha sido modificado extensamente para dar soporte al mecanismo de *Mini-Blockchain*.

III. Configuración de Redes *Mini-Blockchain* Privadas

En esta sección se describirá el proceso de modificación, compilación e instalación de *Cryptonite* para la creación de redes *Mini-Blockchain* privadas en un sistema operativo GNU/Linux, usando como ejemplo la distribución Ubuntu 18.04 LTS.

III-A. Instalación de Dependencias

Según la documentación incluida con el código fuente de *Cryptonite* [22], este sistema requiere que las siguientes bibliotecas estén instaladas en el sistema operativo:

SSL implementación del protocolo TLS de seguridad en capa de transporte del proyecto OpenSSL.
Boost biblioteca de utilerías de propósito general para extender las capacidades del lenguaje C++.
BDB base de datos NoSQL *Berkeley DB* desarrollada por Oracle Corporation.
GMP biblioteca de aritmética entera y punto flotante con precisión arbitraria del proyecto GNU.

Adicionalmente, se requiere de las herramientas `pkg-config` y la suite de utilidades de compilación *Autotools* de GNU. Estas bibliotecas y herramientas se pueden instalar facilmente en Ubuntu 18.04 LTS ejecutando el comando mostrado en el Listado 1 como usuario `root`.

Listado 1: Instalación de dependencias de *Cryptonite*.

```
apt install libgmp-dev libssl-dev \
libboost-all-dev libdb-dev libdb++-dev \
build-essential autotools-dev automake \
autoconf pkg-config
```

III-B. Descarga del Código Fuente

Para compilar e instalar *Cryptonite* es necesario descargar su código fuente el cual se encuentra disponible en [22]. Esta descarga se puede realizar en la terminal con la herramienta de control de versiones `git`, usando el comando mostrado en el Listado 2.

Listado 2: Descarga del código fuente de *Cryptonite*.

```
git clone git@github.com:pallas1/Cryptonite.git
```

La creación de una *Mini-Blockchain* privada con *Cryptonite* consiste en modificar ciertos datos embebidos en el código fuente del sistema que son usados para guiar la generación del bloque génesis[2] de la *Mini-Blockchain*. Así mismo, también es necesario modificar las direcciones de los servidores que el sistema utilizará como semillas DNS durante su procedimiento de inicialización y búsqueda de pares.

III-C. Configuración del Bloque Génesis

Los parámetros a modificar se encuentran todos ubicados en el archivo `chainparams.cpp`, el cual está ubicado dentro de la estructura de directorios del proyecto *Cryptonite* en el subdirectorio `src`.
La modificación de parámetros debe realizarse en dos etapas. En la primera etapa se debe ubicar el miembro `msg` de la clase `CMainParams`, el cual contiene lo indicado en el Listado 3. Este mensaje se debe modificar con un nuevo contenido de selección arbitraria. Por tradición, y siguiendo el ejemplo de Bitcoin [23], se suele utilizar la fecha de generación del bloque génesis[3] junto a un titular de actualidad para que sirvan como marca de tiempo del bloque génesis, aunque como se muestra en el Listado 4 el mensaje a utilizar es completamete arbitrario.

Listado 3: Mensaje inicial de *Cryptonite*.

```
string msg = "2014/07/27 - Epoch Times - How
    Bitcoin Compares...";
```

Listado 4: Mensaje inicial modificado.

```
string msg = "2019/08/06 - What's the most you've
    ever lost on a coin toss?";
```

Luego se debe modificar el campo `genesis.nTime` al cual se le debe colocar la fecha actual en formato *epoch* (época) de UNIX. Para esto se puede utilizar algún servicio como [25] o el comando *date* de la distribución con el argumento "`+ %s`". Finalmente se coloca el valor 0 (cero) en el campo `genesis.nNonce`.
Una vez hechos estos cambios es necesario compilar y ejecutar *Cryptonite* para que este calcule dos parámetros adicionales que son necesarios para la generación correcta del bloque génesis, los cuales son el valor *hash* raíz del árbol de Merkle [26] de transacciones del bloque y el *nonce* (que se acaba de colocar en cero temporalmente) utilizado para poder minar el bloque génesis. Para que *Cryptonite* imprima el *hash* del árbol de transacciones es necesario descomentar la linea mostrada en el Listado 5, pudiendo editarse de forma similar a como muestra el Listado 6 para hacer más sencilla la identificación de dicha salida.

Listado 5: Impresión del *hash* de Merkle.

```
//printf("%s\n", genesis.hashMerkleRoot.ToString()
    .c_str());
```

[2]Se conoce como bloque génesis al primer bloque de una *Blockchain*, siendo este el único que no posee bloques anteriores [23].
[3]Los desarrolladores de *Cryptonite* colocaron la fecha en formato ISO 8601 [24], aunque el formato a utilizar en realidad es completamente arbitrario.

Listado 6: Linea de impresión editada.

```
printf("Hash de la raiz de Merkle: %s\n", genesis.
    hashMerkleRoot.ToString().c_str());
```

Así mismo, es necesario comentar la linea mostrada en el Listado 7 para evitar una validación de seguridad sobre el *hash* del árbol de transacciones. De no realizar este paso, *Cryptonite* fallará antes de poder calcular los nuevos valores para el bloque génesis.

Listado 7: Validación del árbol de transacciones.

```
assert(genesis.hashMerkleRoot == uint256("0
x9dabd47e692ed615eae95da0c95f195a7dea9428bb
9f9e0cd4c7d12533bf3667"));
```

En este punto se procede a compilar e instalar *Cryptonite*, ejecutando los comandos mostrados en el listado 8. Estos comandos son estándares para compilar proyectos gestionados por el sistema de construcción de GNU *Autotools*. Sin embargo, el parámetro pasado al comando `./configure` merece ser revisado. Este parámetro sirve para forzar a que el proceso de compilación utilice la versión de *Berkeley DB* instalada en el sistema; de no utilizarlo, la compilación fallaría porque *Cryptonite* espera utlizar una versión específica de *Berkeley DB*

Listado 8: Compilación e instalación de *Cryptonite*.

```
./autogen.sh
./configure --with-incompatible-bdb
make
sudo make install
cryptonited
```

La ejecución del demonio de *Cryptonite* como el último comando del Listado 8 provocará que el sistema comience el proceso de minado del bloque génesis. Cuando este proceso culmine, el demonio de *Cryptonite* emitirá una salida similar a la mostrada en la Figura 1, donde se puede apreciar el valor del *hash* del árbol de transacciones en la primera linea, y el valor del *nonce* del bloque en la última linea, que comienza con el texto "*Found Genesis*" (Génesis encontrado).
Finalmente, se debe tomar el *nonce* encontrado y colocarlo como valor del campo `genesis.nNonce`; el *hash* del árbol de transacciones se debe colocar en la verificación mostrada en el Listado 7 que fue comentada previamente y ahora debe ser descomentada de nuevo; y el valor del *hash* del bloque debe ser colocado dentro de la comparación que se muestra en el Listado 9. Hecho esto, es necesario entonces el volver a compilar el proyecto con los comandos del Listado 8, pero sin ejecutar el demonio de *Cryptonite*.

Listado 9: Verificación del *hash* del bloque génesis.

```
if(hashGenesisBlock != uint256("0x000009a46
0ccc429ac6e53c91c6ed2d96697884b8b656a903042
faff8971c5aa"))
```

Para poder ejecutar el demonio de *Cryptonite* correctamente es necesario crear un archivo llamado `cryptonite.conf` dentro del directorio `/.cryptonite/`, el cual debe contener los parámetros mostrados en el Listado 10. Los valores de ambos campos del archivo son de selección arbitraria, los valores que se muestran en el Listado 10 son solamente referenciales.

Listado 10: Compilación e instalación de *Cryptonite*.

```
rpcuser=antonchigurh
rpcpassword=VGhlIGNvaW4gZG9uJ3QgaGF2ZSBubyBzYXkK
```

III-D. Configuración de las Semillas DNS

Al ser un sistema de red *Peer-to-Peer*, *Cryptonite* necesita realizar un proceso de descubrimiento de pares al momento de inicializarce para poder sincronizar el estado de la *Mini-Blockchain* local con la red. De la misma forma que *Bitcoin*, *Cryptonite* utiliza una estrategia de semillas DNS, mediante la cual ciertos pares conocidos actuarán como los primeros pares a los cuales se podrá conectar el nodo *Cryptonite*. Por defecto *Cryptonite* utiliza las siguientes semillas DNS:

- explorer.cryptonite.info
- xcn.suprnova.cc
- explorer.digicent.org

Dado que estas semillas DNS están sincronizadas a la red principal de *Cryptonite* en lugar de a la red privada configurada en la Sección III-C, estas no servirán como pares de inicialización. Para modificar estas semillas es necesario editar las lineas mostradas en el Listado 11, sustituyendo las direcciones DNS mostradas por direcciones DNS que apunten a nodos sincronizados con la red privada. Estas direcciones se pueden configurar en un servidor DNS en la red local, como por ejemplo Bind 9 [27].

Listado 11: Configuración de las semillas DNS.

```
vSeeds.push_back(CDNSSeedData(
    "explorer.cryptonite.info",
    "explorer.cryptonite.info"));
vSeeds.push_back(CDNSSeedData(
    "xcn.suprnova.cc",
    "xcn.suprnova.cc"));
vSeeds.push_back(CDNSSeedData(
    "explorer.digicent.org",
    "explorer.digicent.org"));
```

III-E. Instalación y Configuración de un Minero M7

Para poder realizar el minado de la Mini-Blockchain privada es necesario instalar y configurar un proceso minero con soporte para el algoritmo de prueba de trabajo M7 usado por *Cryptonite*. Un ejemplo de un minero factible se encuentra disponible en el repositorio [28], donde el grupo de desarrolladores de *Cryptonite* provee una versión modificada del minero de Bitcoin *cpuminer* [29] adaptado para dar soporte al algoritmo M7.
De la misma manera en que se hizo con el sistema *Cryptonite*, la configuración de *minerd* comienza por la instalación de sus dependencias. Según la documentación del código fuente de

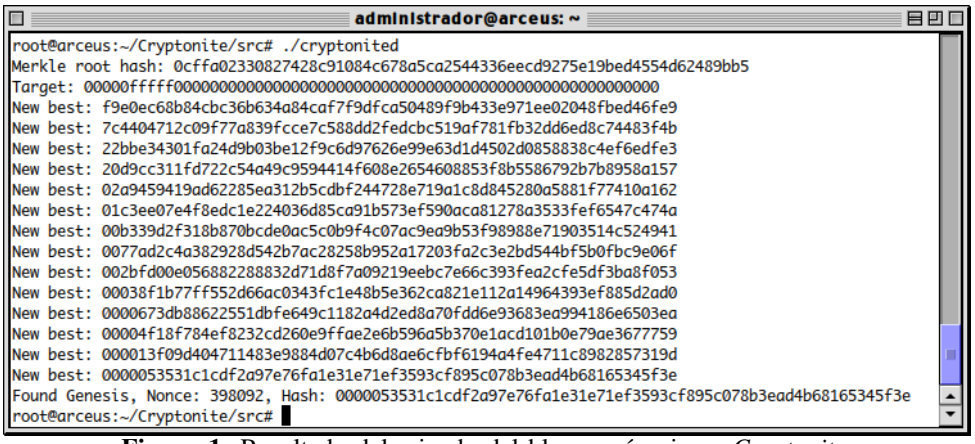

Figura 1: Resultado del minado del bloque génesis en *Cryptonite*.

minerd, este tiene las siguientes tres dependencias, que pueden ser instaladas en Ubuntu 18.04 LTS con el comando mostrado en el Listado 12.

cURL programa y biblioteca de conexión multi-protocolo para capas de transporte y aplicación.

OpenSSL kit de herramientas y bibliotecas para conexión segura con el protocolo SSL/TLS.

Jansson biblioteca para el lenguaje C que permite manipulación de datos en formato JSON.

Listado 12: Instalación de dependencias de *minerd*.

```
apt install libcurl4-openssl-dev libssl1.0-dev\
libjansson-dev
```

Una vez instaladas las dependencias, se debe descargar el código fuente de *minerd* del repositorio disponible en [28] usando el comando mostrado en el Listado 13.

Listado 13: Descarga del código fuente de *minerd*.

```
git clone git@github.com:MiniblockchainProject/
    Minerd.git
```

La configuración y compilación del minero sigue los pasos usuales para la compilación de programas gestionados por las herramientas de GNU *Autotools* como se ve en el Listado 14. Las lineas iniciadas por `CFLAGS` y `CXXFLAGS` en el Listado 14 establecen las opciones de optimización de código del compilador GCC para la compilación del proyecto.

Listado 14: Compilación de *minerd*.

```
./autogen.sh
CFLAGS="-O3 -march=native -mtune=native" \
CXXFLAGS="-O3 -march=native -mtune=native" \
./configure
make
make install
```

Finalmente, para iniciar el minero *minerd* se ejecuta el comando mostrado en el Listado 15. Como se observa, los parámetros que puede recibir *minerd* son:

url dirección Web donde se encuentra algún nodo de la red privada ejecutándose en modo servidor.[4].

user el usuario RPC que se configuró en el Listado 10.

pass *password* del usuario RPC, configurado en el Listado 10.

threads cantidad de hilos de ejecución a utilizar en el proceso de minado.

Listado 15: Ejecución de *minerd*.

```
minerd --url http://127.0.0.1:8252 \
--user antonchigurh \
--pass "VGhlIGNvaW4gZG9uJ3QgaGF2ZSBubyBzYXkK" \
--threads 2
```

Si el minero se ejecutó correctamente y logró conectarse con el nodo *Cryptonite* especificado, entonces debería mostrar una salida similar a la que se ve en la Figura 2, una vez que *minerd* comienze a minar.

III-F. Configuración de un Servicio en Linux

Configurar servicios de sistema para los nodos de la red privada es un procedimiento completamente opcional. Sin embargo, el aplicar estas configuraciones permite que tanto el nodo *Cryptonite* como el minero inicien automáticamente al arrancar el sistema operativo, en lugar de tener que ser iniciados manualmente.

III-F1. Servicio para el nodo cryptonited: La configuración de un servicio de sistema en sistemas operativos Linux basados en la distribución Debian y sus derivados como lo es el sistema operativo Ubuntu 18.04 LTS consiste en la preparación de un archivo de control para *Systemd*, que es el sistema *init*[5] de la distribución. Para el caso de *Cryptonite*, un archivo de configuración de *systemd* como el que se ve en el Listado 16 es apropiado. Este archivo debe ser colocado en el directorio `/lib/systemd/system` con el nombre `cryptonite.service`.

[4]Es necesario especificar el puerto RPC del nodo. Por defecto, *Cryptonite* usa el puerto 8252 como puerto RPC.

[5]En terminología de UNIX *init* es el primer proceso iniciado por el sistema operativo [30]. *Systemd* es una implementación de dicho proceso para sistemas GNU/Linux, el cual incorpora un *framework* de gestión de servicios de sistema.

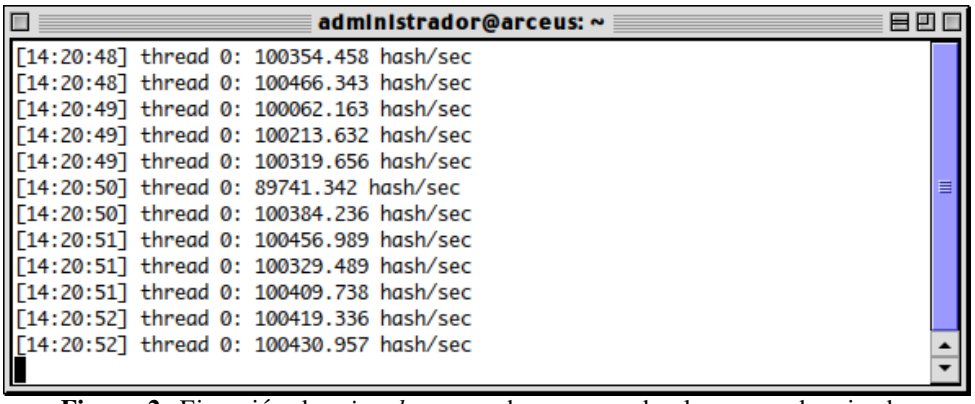

Figura 2: Ejecución de *minerd* una vez ha comenzado el proceso de minado.

Listado 16: Definición de servicio para *cryptonited*.

```
[Unit]
Description=Cryptonite node daemon
[Service]
Type=forking
User=root
Group=root
WorkingDirectory=/root
PIDFile=/root/.cryptonite/cryptonited.pid
ExecStart=/usr/bin/cryptonited -server -daemon
[Install]
WantedBy=multi-user.target
```

III-F2. Servicio para el minero minerd: Posteriormente se debe crear un archivo de configuración similar para el proceso *minerd*, el cual se puede observar en el Listado 17. Este archivo debe ser igualmente copiado en el directorio `/lib/systemd/system`, dándosele el nombre `miner.service`.

Listado 17: Definición de servicio para *minerd*.

```
[Unit]
Description=Minerd service
After=cryptonite.service
[Service]
Type=forking
PIDFile=/var/run/minerd.pid
ExecStartPre=/bin/rm -f /var/run/minerd.pid
ExecStart=/usr/bin/minerd_daemon
[Install]
WantedBy=multi-user.target
```

Como *minerd* no posee la capacidad de ejecutarse como un proceso demonio de UNIX, es necesario crear un programa adicional que se ejecute como un demonio primero para luego ejecutar el minero. Esto se puede lograr utilizando un pequeño programa como el mostrado en el Listado 18, el cual se encarga de convertirse en un proceso demonio mediante la llamada al sistema `daemon` de Linux. Si la llamada tiene éxito, entonces el programa ejecuta el minero *minerd* mediante la llamada al sistema `execlp`, usando los mismos parámetros mostados en el Listado 15. Para que el programa del Listado 18 funcione con el servicio definido en el Listado 17, debe ser compilado con el nombre `minerd_daemon` y copiado en el directorio `/usr/bin/`.

Listado 18: Ejecución de *minerd* como un demonio.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
  /* Turn the process into a daemon. */
  int rc = daemon(1, 1);
  if (rc) {
    perror("Failed to daemonize.");
    return EXIT_FAILURE;
  }

  /* Create the PID file. */
  FILE * pid = fopen("/var/run/minerd.pid", "w");

  if (pid == NULL) {
    perror("Failed to create PID file.");
    return EXIT_FAILURE;
  }

  fprintf(pid, "%d", getpid());
  fclose(pid);

  /* Execute minerd. */
  rc = execlp("/usr/bin/minerd", "/usr/bin/minerd"
      , "--url", "http://127.0.0.1:8252", "--user"
      , "antonchigurh", "--pass", "
      VGhlIGNvaW4gZG9uJ3QgaGF2ZSBubyBzYXkK", "--
      threads", "2", NULL);
  if (rc) {
    perror("Failed to execute minerd.");
    return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

IV. Conclusiones y Trabajos Futuros

En este trabajo se presentó un manual de configuración del sistema de Mini-Blockchain *Cryptonite* que indica los pasos a seguir para adaptar el código fuente del sistema para configurar una red privada. Las redes privadas creadas mediante este método pueden ser usadas para el desarrollo de extensiones y/o modificaciones al sistema *Cryptonite*, tales como las planteadas en el proyecto propuesto en [11].

La creación de redes privadas sirve adicionalmente para la definición de entornos de desarrollo para aplicaciones sobre

una criptomoneda. En efecto, tener la posibilidad de crear redes privadas permitirá en un futuro el poder desarrollar y depurar contratos inteligentes para la Mini-Blockchain *Cryptonite* de una forma similar a como se hace con la criptomoneda *Ethereum* [31].

Sin embargo, el procedimiento planteado en la Sección III es completamente manual. En base a esto se proponen como trabajos futuros los siguientes:

- La automatización del proceso de configuración de la red privada mediante el uso de parches y programas *shell script*.
- Integrar la automatización del proceso al sistema de construcción basado en GNU *Autotools* de *Cryptonite*.

REFERENCIAS

[1] S. Nakamoto, "*Bitcoin: A peer-to-peer electronic cash system*," https://nakamotoinstitute.org/bitcoin/, 2008, revisado el: 27-07-2019.

[2] F. Tschorsch y B. Scheuermann, "*Bitcoin and beyond: A technical survey on decentralized digital currencies*," IEEE Communications Surveys & Tutorials, vol. 18, no. 3, pp. 2084–2123, 2016.

[3] I.-C. Lin y T.-C. Liao, "*A survey of blockchain security issues and challenges.*" IJ Network Security, vol. 19, no. 5, pp. 653–659, 2017.

[4] J. Yli-Huumo, D. Ko, S. Choi, S. Park, y K. Smolander, "*Where is current research on blockchain technology?—a systematic review*," PloS one, vol. 11, no. 10, p. e0163477, 2016.

[5] J. A. Bergstra y K. de Leeuw, "*Questions related to bitcoin and other informational money*," arXiv preprint arXiv:1305.5956, 2013.

[6] J. Bruce, "*Purely p2p crypto-currency with finite mini-blockchain*," https://bitcointalk.org/index.php?topic=195275.0, 2013, revisado el: 27-07-2019.

[7] A. Wagner, "*Ensuring network scalibility: How to fight blockchain bloat*," Bitcoin Magazine, vol. 6, 2014.

[8] J. Bruce, "*The mini-blockchain scheme, rev. 3*," http://cryptonite.info, 2017, revisado el: 27-07-2019.

[9] V. Buterin *et al.*, "*A next-generation smart contract and decentralized application platform*," white paper, 2014.

[10] G. Wood, "*Ethereum: A secure decentralised generalised transaction ledger*," Ethereum Project Yellow Paper, 2014.

[11] C. Sanguña, S. Suarez, A. Russoniello, M. A. Astor, y W. Pereira, "*Proposals for a smart contracts platform for cryptocurrencies based on the mini-blockchain scheme*," en Memorias de la Sexta Conferencia Nacional de Computación, Informática y Sistemas CoNCISa. Mérida, Venezuela: Sociedad Venezolana de Computación, noviembre 2018, pp. 88–92.

[12] The Mini-blockchain Project, "*Cryptonite - mini-blockchain project*," http://cryptonite.info/, revisado el: 25-07-2019.

[13] W. Dai, "*b-money*," http://www.weidai.com/bmoney.txt, 1998, revisado el: 27-07-2019.

[14] N. Szabo, "*Secure property titles with owner authority*," http://nakamotoinstitute.org/secure-property-titles/, 1998, revisado el: 27-07-2019.

[15] N. Szabo, "*Bit gold*," http://nakamotoinstitute.org/bit-gold/, 2005, revisado el: 27-07-2019.

[16] W. F. Ehrsam, C. H. Meyer, J. L. Smith, y W. L. Tuchman, "*Message verification and transmission error detection by block chaining*," Estados Unidos de América Patente 4 074 066, febrero 14, 1978.

[17] A. Back, "*Hashcash*," http://www.cypherspace.org/hashcash/, 1997, revisado el: 27-07-2019.

[18] A. Back, "*Hashcash – a denial of service counter-measure*," http://www.hashcash.org/papers/hashcash.pdf, 2002, revisado el: 27-07-2019.

[19] C. Dwork y M. Naor, "*Pricing via processing or combatting junk mail*," en Annual International Cryptology Conference. Springer, 1992, pp. 139–147.

[20] A. Reiner, "*Ultimate blockchain compression*," https://bitcointalk.org/index.php?topic=88208, 2012, revisado el: 27-07-2019.

[21] J. D. Bruce, "*The mini-blockchain scheme, rev. 2*," https://cryptonite.info/files/mbc-scheme-rev2.pdf, 2014, revisado el: 27-07-2019.

[22] Pallas1, "*Cryptonite core integration/staging tree*," https://github.com/pallas1/Cryptonite, revisado el: 25-07-2019.

[23] C. Barski y C. Wilmer, *Bitcoin for the Befuddled*. No starch press, 2014.

[24] ISO/TC 154, "*Date and time – representations for information interchange – part 1: Basic rules*," International Organization for Standardization, Ginebra, Suiza, Estándar ISO 8601-1:2019, febrero 2019.

[25] M. O. de Coul, "*Epoch & unix timestamp conversion tools*," https://www.epochconverter.com/, revisado el: 25-07-2019.

[26] R. C. Merkle, "*A digital signature based on a conventional encryption function*," en Conference on the theory and application of cryptographic techniques. Springer, 1987, pp. 369–378.

[27] Internet Systems Consortium, "*Bind 9 versatile, classic, complete name server software*," https://www.isc.org/bind/, revisado el: 25-07-2019.

[28] Miniblockchain Project, "*minerd*," https://github.com/MiniblockchainProject/Minerd, revisado el: 08-08-2019.

[29] J. Garzik, "*Cpu miner for bitcoin*," https://github.com/jgarzik/cpuminer, revisado el: 08-08-2019.

[30] A. Tanenbaum, W. Stallings, y M. Kerrish, *Sistemas Operativos modernos., 3ra edición.* México: Pearson, 2009.

[31] A. Antonopoulos y G. Wood, *Mastering Ethereum: Implementing Digital Contracts and DApps*. O'Reilly Media, 2018.