TP Blockchain

Jérémy Briffaut

March 18, 2024

1 Objectifs

MultiChain is an open source platform for private blockchains, which offers a rich set of features including extensive configurability, rapid deployment, permissions management, native assets and data streams. Although it is designed to enable private blockchains, MultiChain provides maximal compatibility with the bitcoin ecosystem, including the peer-to-peer protocol, transaction/block formats and Bitcoin Core APIs/runtime parameters.

Lors de ce TP, vous verrons :

- Déploiement d'un environnement blockchain privé sur 3 machines virtuelles
- Installation de multichain
- Prise en main de multichain
- Blockchain en mode flux de données
- Installation d'un frontend Web
- Gestion des autorisations
- Chiffrement de flux

Ce TP est notamment basé sur la documentation officielle de multichain disponible sur `http://www.multichain.com`

2 Mise en place de l'environnement

Nous allons utiliser la distribution Centos version 7 pour déployer notre environnement. Commencez par copier, puis décompresser l'image VMware `/mnt/nfs/IMAGES/Centos7-TP-blockchain-2023.zip` dans `/user/user/`, l'ouvrir avec vmware.
Clonez votre machine 2 fois (*link clone*) (bien stocker votre clone dans `/home/user/`) afin

d'avoir 3 machines. Nommez les `miner-admin, miner-slave01, miner-slave02`.

Démarrez les 3 machines et vérifiez que vous avez bien du reseau sur chacune :

```
ping␣google.fr
```

Dans le cas contraire, voici comment ré-initialiser le réseau

```
pkill␣udevd
rm␣/etc/udev/rules.d/70*net*
reboot
```

Centos utilise `yum` comme gestionnaire de paquetage, pour installer vim :

```
yum␣update
yum␣install␣vim
```

Vous avez maintenant 3 machines virtuelles. Loggez vous sur les deux machines, puis récupérer les adresses IPs des deux machines :

- `miner-admin` :
- `miner-slave01` :
- `miner-slave02` :

Lancer maintenant 3 terminal sur votre machine hôte et connectez vous en ssh sur les 3 machines (souvenez vous a quelle machine correspond quel terminal pour la suite).

```
ssh␣root@IP_VM
#mot␣de␣passe␣:␣azerty

#pour␣renommer␣le␣prompt␣de␣votre␣terminal␣:
export␣PS1="miner-admin|$PS1"
```

Vos machines doivent être synchronisé au niveau temps, sur chacune d'elle, entrez :

```
yum␣install␣ntpdate␣-y
ntpdate␣pool.ntp.org
date
```

3 Installation du noeud Miner-Admin

Cette machine sera notre machine administrateur de la blockchain.

3.1 Installation de multichain

Multichain est une implémentation open-source d'une blockchain compatible bitcoin (`http://www.multichain.com/`)

```
yum␣install␣-y␣wget␣vim␣htop
cd␣/tmp
wget␣http://www.multichain.com/download/multichain1.0alpha28.tar.gz
tar␣xvzf␣multichain1.0alpha28.tar.gz
cd␣multichain1.0alpha28
mv␣multichaind␣multichain-cli␣multichain-util␣/usr/local/bin
```

3.2 Installation des noeuds secondaires

Sur chacun des noeuds :

```
yum␣install␣-y␣wget␣vim␣htop
cd␣/tmp
wget␣https://www.multichain.com/download/multichain-2.3.3.tar.gz
tar␣-xvzf␣multichain-2.3.3.tar.gz
cd␣multichain-2.3.3
mv␣multichaind␣multichain-cli␣multichain-util␣/usr/local/bin
```

4 Prise en main de multichain

4.1 Creation d'une blockchain

D'abord nous créerons un nouveau blockchain nommé `chain1`. Sur le noeud `miner-admin`, exécutez cette commande :

```
multichain-util␣create␣chain1
```

Visualisez les paramètres par défaut de la blockchain (ceux-ci peuvent être modifiés mais nous utilisons ceux par défaut pour l'instant) :

```
cat␣~/.multichain/chain1/params.dat
```

Initialisez la blockchain, y compris le minage du bloc "genesis" :

```
multichaind␣chain1␣-daemon
```

Vous devriez être informé que le service a démarré et ensuite que le bloc de genèse a été trouvé. L'adresse pour se connecter a votre chaine et du type :

```
chain1@[ip-address]:[port]
```

Notez la pour la suite.

Votre chaîne est stocker dans :

```
ls␣~/.multichain/chain1/
```

Vous pouvez suivre l'avancement du minage via :

```
tail␣~/.multichain/chain1/debug.log
```

4.2 Se connecter a la blockchain

Nous allons maintenant connecter le noeud `miner-slave01` à notre chaine `chain1`. Pour cela entrez cette commande sur le noeud `miner-slave01` :

```
multichaind␣chain1@[ip-address]:[port]
```

Vous pouvez remarquer que la connexion échoue. Il faut tout d'abord ouvrir e port correspondant sur le noeud `miner-admin`. Entrer donc cette commande sur ce noeud (remplacer 4353 par le port correspondant a l'adresse de votre chaine) :

```
firewall-cmd␣--permanent␣--zone=public␣--add-port=4353/tcp
systemctl␣restart␣firewalld.service
```

Essayer ensuite de vous reconnecter depuis le noeud `miner-slave01` :

```
multichaind␣chain1@[ip-address]:[port]

#[...]
#Retrieving␣blockchain␣parameters␣from␣the␣seed␣node␣172.16.96.141:4353␣...
#Blockchain␣successfully␣initialized.
#[...]
```

Cette commande doit vous retourner que la blockchain a été avec succès initialisé, mais que vous n'avez pas la permission de vous connecter. Vous devez aussi voir un message contenant l'adresse (identifiant) de votre noeud. Cette adresse correspond a votre *wallet address*. Copier cette adresse pour la suite :

```
1TDpFJ5c7heD5SrGR.....
```

Revenez sur votre noeud `miner-admin` et donnez les droits de connexion à votre noeud `miner-slave01` :

```
multichain-cli␣chain1␣grant␣1TDpFJ5c7heD5SrGR.....␣connect
```

Essayez maintenant de vous connecter de nouveau depuis `miner-slave01` :

```
multichaind␣chain1␣-daemon
```

Vous devriez être averti que le noeud a été démarré, et il devrait afficher adresse de ce la deuxième noeud.

Verifiez les traces de votre nouveau noeud :

```
tail␣~/.multichain/chain1/debug.log
```

4.3 Quelques commandes en mode interactif

Pour entrer en mode interactif, entrez cette commande sur les deux noeuds :

```
multichain-cli␣chain1
```

Mainteannt que vos deux noeuds sont interconnectés, vous pouvez entrer ces commandes sur l'un ou l'autre.

Pour obtenir des informations sur votre chaine :

```
getinfo
```

Pour voir la liste des commandes disponibles :

```
help
```

Pour afficher la liste des permissions (de chaque adresse) :

```
listpermissions
```

Pour créer une nouvelle adresse dans le wallet (pour votre noeud):

```
getnewaddress
```

Pour lister les adresses de votre wallet (pour votre noeud):

```
getaddresses
```

Récupérer les paramètres de votre blockchain (en se basant sur le fichier params.dat):

```
getblockchainparams
```

Pour chaque n oeud, récupérer la liste des noeuds auquel il est connectés:

```
getpeerinfo
```

4.4 Petit exercice

Connecter votre noeud `miner-slave02` a votre noeud `miner-slave01`. Vérifier ensuite sur chaque noeud avec la commande `getpeerinfo`.

5 Multichain en mode flux de données

5.1 Les Streams

Créez un stream, qui peut être utilisé pour le stockage et la récupération de données "générales". Sur le noeud admin, exécutez la commande suivante :

```
create␣stream␣stream1␣false
```

Le paramètre `false` indique que le flux ne peut être écrits que par ceux ayant des permissions explicites. Voyons ses permissions :

```
listpermissions␣stream1.*
```

Ainsi pour l'instant, seul le noeud admin a la capacité d'écrire dans ce stream et de l'administrer. Publiez quelque chose avec la clé key1 :

```
publish␣stream1␣key1␣73747265616d2064617461
```

Le txid de l'élément du stream est retourné. Vérifiez maintenant que le stream est visible sur l'autre noeud. Sur le deuxième noeud :

```
liststreams
```

(Le stream root est créer par défaut dans la blockchain.)

Maintenant nous voulons que le deuxième noeud souscrive au stream, puis visualisez son contenu :

```
subscribe␣stream1
liststreamitems␣stream1
```

Maintenant ajoutez au deuxième noeud le droit de publier dans le stream. Sur le premier noeud :

```
grant␣1.adresse..du..portefeuille..de..miner−slave01..␣receive,send
grant␣1.adresse..du..portefeuille..de..miner−slave01..␣stream1.write
```

Notez que l'adresse a besoin des deux permissions générales "émetteuses-réceptrices" pour la blockchain, aussi bien que la permission d'écrire dans le stream spécifique.

Maintenant, sur le second noeud, publier de nouvelles donnée dans ce stream :

```
publish␣stream1␣key1␣736f6d65206f746865722064617461
publish␣stream1␣key2␣53747265616d732052756c6521
```

Maintenant examinez le contenu du stream de différentes manières. Sur le premier noeud :

```
subscribe␣stream1
liststreamitems␣stream1␣#␣(should␣show␣3␣items)
liststreamkeys␣stream1␣#␣(2␣keys)
```

```
liststreamkeyitems␣stream1␣key1␣#␣(2␣items␣with␣this␣key)
liststreampublishers␣stream1␣#␣(2␣publishers)
liststreampublisheritems␣stream1␣1.adresse..du..portefeuille..de..miner-slave01..␣#␣(2␣
    items␣by␣this␣publisher)
```

5.2 Petit exercice

Ajouter votre noeud `miner-slave02` a votre stream, et publiez des données depuis ce noeud. Vérifiez sur les autres que les données apparaissent.

Créer un script/programme pour automatiser la publication/récupération des données d'un stream.

6 Installation d'un frontend Web

6.1 multichain-explorer

Nous allons installer sur le noeud admin un premier frontend Web, multichain-explorer, qui permet d'interagir "graphiquement" avec votre blockchain :

Commençons l'installation :

```
cd␣/root
yum␣install␣epel-release␣-y
yum-config-manager␣--enable␣epel
yum␣update
yum␣install␣python-pip␣-y
yum␣install␣sqlite-devel␣-y
yum␣install␣python-devel␣-y
yum␣install␣gcc␣pip␣-y
pip␣install␣pycrypto
pip␣install␣py-ubjson
wget␣https://github.com/MultiChain/multichain-explorer/archive/master.zip
unzip␣master.zip
cd␣multichain-explorer-master/
python␣setup.py␣install
```

Récupérer le port RPC utilisé par votre blockchain :

```
cat␣~/.multichain/chain1/params.dat␣|␣grep␣rpc
```

Puis paramétrez pour votre blockchain `chain1` :

```
#␣remplacer␣2754␣par␣votre␣port␣RPC
echo␣"rpcport=2754"␣>>␣/root/.multichain/chain1/multichain.conf
cp␣chain1.example.conf␣chain1.conf
sed␣-i␣"s/^port␣.*/port␣8080/"␣chain1.conf
```

```
sed␣-i␣"s/^host␣.*/host␣0.0.0.0/"␣chain1.conf
```

Ajoutez l'accès au ports Web dans votre pare-feu

```
firewall-cmd␣--permanent␣--zone=public␣--add-port=8080/tcp
firewall-cmd␣--permanent␣--zone=public␣--add-port=80/tcp
systemctl␣restart␣firewalld.service
```

Lancez le serveur web de votre frontend

```
python␣-m␣Mce.abe␣--config␣chain1.conf␣--commit-bytes␣100000␣--no-serve
python␣-m␣Mce.abe␣--config␣chain1.conf
```

Connectez vous avec le navigateur de votre machine hôte sur le frontend :

```
http://IP_VM_ADMIN:8080
```

6.2 Petit exercice

Prenez le temps de découvrir l'interface, notamment :

- Lister les dernières transactions
- Qu'est-ce qu'une confirmation ?
- Que contient une transaction ?
- Quelle type de transaction apparaissent ?
- Lister les miners, les noeuds, les assets, les noeuds
- Lister les inputs/outputs
- Etudiez les différents formats JSON/HEX
- Afficher le hash de chaque block, quelle est la difficulté ici (nombre de 0 minimum débutant un hash) ?

7 Gestion des Autorisations

7.1 MultiChain permissions management

MultiChain fournit un contrôle total des autorisations au niveau du réseau.

7.2 Autorisations dans MultiChain

Dans MultiChain, il existe huit types d'autorisations globales pouvant être accordées aux adresses:

- connect : Pour se connecter à d'autres noeuds et voir le contenu de la blockchain.
- send : Envoyer des fonds, c'est-à-dire signer les entrées de transactions.
- receive : Pour recevoir des fonds, c'est-à-dire apparaître dans les résultats des transactions.
- issue : Pour émettre des actifs, c'est-à-dire signer des entrées de transactions qui créent de nouveaux actifs natifs.
- create : Pour créer des flux, c'est-à-dire signer des entrées de transactions qui créent de nouveaux flux.
- mine : Minage, c'est-à-dire signer les métadonnées des transactions.
- activate : Pour changer les autorisations de connexion, d'envoi et de réception pour d'autres utilisateurs, c'est-à-dire signer des transactions qui modifient ces autorisations.
- admin : Pour changer toutes les autorisations pour les autres utilisateurs, y compris issue, mine, activate et admin.

Toutes les autorisations sont attribuées en fonction de l'adresse, où les adresses peuvent être soit des hachages de clés publiques ou soit des hachages de script. Selon les paramètres de `anyone-can-*` dans les paramètres de la blockchain (fichier `params.dat`), certaines opérations peuvent être totalement illimitées. Les autorisations peuvent également être rendues temporaires en les limitant à une gamme spécifique de nombres de blocs, de sorte qu'ils ne soient disponibles que pour les transactions qui apparaissent dans ces blocs.

Toutes les autorisations sont accordées et révoquées à l'aide de transactions réseau contenant des métadonnées spéciales, qui sont faciles à envoyer en utilisant les commandes de `grant` ou `revoke` pour `multichain-cli` ou l'API JSON-RPC. Le mineur du premier bloc de genèse d'une chaîne reçoit automatiquement toutes les autorisations. Cet administrateur peut alors accorder des autorisations à d'autres adresses, y compris les `admin` et `activate`.

Si un wallet MultiChain contient plusieurs paires de clés public / privé, lors de l'exécution d'une opération restreinte telle que l'émission d'actifs ou le minage, il utilisera automatiquement l'une de ses clés (et les sorties de transactions non utilisées correspondantes) autorisées à effectuer cette opération. Il existe également possible de contrôler l'adresse utilisée.

MultiChain permet également d'autoriser les autorisations d'administration et d'émission pour les actifs individuels, et d'administrer, d'activer et d'écrire des autorisations pour les flux.

7.3 Comment les autorisations sont appliquées

L'autorisation `connect` est appliquée lors de la prise de contact pair à pair qui se déroule entre les noeuds, dans lequel chaque noeud prouve sa possession de la clé privée correspondant à une adresse particulière. Si l'autorisation de se connecter au réseau est révoquée à partir d'une

adresse, les noeuds déconnectent immédiatement toutes les connexions avec les noeuds qui ont utilisé l'adresse révoquée lors de la prise de contact.

D'autres autorisations sont appliquées en vérifiant si chaque transaction (y compris les transactions d'attribution d'autorisation) est valide en fonction des transactions d'autorisations qui lui ont été soumises précédemment. (Une exception concerne les autorisations de réception, qui peuvent être utilisées dans la même transaction que leur attribution). Bien que différents noeuds puissent voir des transactions dans des commandes différentes, la chaîne de blocs finalise leur commande définitive et toutes les transactions sont "rejouées" dans l'ordre par tous les noeuds pour vérifier leur validité. Si une transaction dans un bloc n'est pas autorisée conformément à cette règle, le bloc entier est invalide.

7.4 Autorisations dans les transactions régulières

7.4.1 Transaction régulière

Une transaction régulière, qui ne crée pas d'asset ou d'attribution d'autorisations, est valide si toutes ses entrées et toutes ses sorties sont autorisées.

Chaque entrée est autorisée dans les conditions suivantes: Si `anyone-can-send` est vrai dans les paramètres de la blockchain. Pour une signature régulière, l'adresse correspondant à la pubkey doit avoir des autorisations d'envoi.

Pour les transactions régulières, chaque sortie est autorisée dans les conditions suivantes: Si `anyone-can-receive` est vrai dans les paramètres de la blockchain.
Si `anyone-can-receive-empty` est vrai dans les paramètres de la blockchain, et la sortie ne contient aucune devise, actif ou autre métadonnée native. Pour un script pubkey, l'adresse de destination (ou l'adresse correspondant à la pub) doit avoir des autorisations de réception.

7.4.2 Autorisations pour d'autres transactions

Une transaction qui crée un nouvel élément est valide si l'une de ses entrées est signée à l'aide d'une adresse avec des autorisations d'émission et le type de signature est $SIGHASH\ _ALL$, ce qui signifie que la transaction entière a été signée. Ces adresses reçoivent automatiquement les autorisations d'administration et d'émission pour le nouvel élément.

Une transaction qui crée un nouveau flux est valide si une ou plusieurs de ses entrées sont signées à l'aide d'une adresse avec des autorisations de création, où la signature couvre la sortie $OP\_DROP + OP\_RETURN$ dans laquelle le flux est créé. Ces adresses reçoivent automatiquement les autorisations d'administration, d'activation et d'écriture pour le nouveau flux.

8 Confidentialité des Flux

8.1 Comment révéler sélectivement les données sur une chaîne de blocs

Par conception, toute donnée brute (raw data) stockée sur une blockchain est visible par chaque noeud connecté à la chaîne. Dans de nombreux cas, cette perte de confidentialité pose un problème important. Dans cette partie, nous allons étudier une approche commune pour résoudre le

problème de confidentialité, où les données sont chiffrées avant d'être stockées et timestampées sur la chaîne. Le mot de passe pour la lecture des données chiffrées est uniquement mis à la disposition d'un sous-ensemble de participants, ce qui laisse les autres incapables de le lire les données chiffrées. En utilisant une combinaison de cryptographie symétrique et asymétrique, nous pouvons le faire de manière efficace.

La méthode utilise trois flux (stream) dont les objectifs sont les suivants:

- Un flux, que nous appelons `pubkeys`, est utilisé par les participants pour distribuer leurs clés publiques (RSA).
- Un second flux, que nous appelons `items`, sert à publier des données de grande taille, dont chacune est cryptée (cryptographie symétrique AES).
- Un troisième flux, que nous appelons `access`, fournit un accès aux données. Pour chaque participant qui devrait voir un jeu de données, une entrée de flux est créée qui contient le mot de passe secret de ces données, crypté à l'aide de la clé publique de ce participant.

Le tutoriel nécessite deux noeuds exécutant Linux et le shell bash, ainsi que `openssl` et `xxd` que vous devez installer au besoin.

8.2 Création des streams

Sur le premier noeud, créez les trois flux comme suit :

```
multichain-cli␣chain1␣create␣stream␣pubkeys␣true
multichain-cli␣chain1␣create␣stream␣items␣true
multichain-cli␣chain1␣create␣stream␣access␣true
```

Chaque commande doit renvoyer une ID de transaction hexadécimale à 64 caractères, identifiant la transaction qui a créé le flux. Les valeurs vraies signifient que les flux sont ouverts en écriture à un participant qui possède des autorisations d'envoi globales pour la chaîne. Vous pouvez afficher ces nouveaux flux, ainsi que tous les autres déjà créés précédemment via :

```
multichain-cli␣chain1␣liststreams
```

8.3 Publish an RSA key pair

Dans cette partie, le premier noeud lira le contenu confidentiel écrit par le second noeud. Dans un premier temps, nous générerons une clé publique RSA sur le premier noeud et publierons la clé publique dans le flux pour que le deuxième noeud puisse la lire.

Sur le premier noeud, trouvez une adresse locale qui a l'autorisation d'envoyer (`send`) en exécutant ces deux commandes:

```
multichain-cli␣chain1␣listpermissions␣send
multichain-cli␣chain1␣listaddresses
```

Une adresse est appropriée si elle apparaît dans la sortie de la première commande, et également dans la deuxième commande avec `"ismine": true`.

Copiez l'adresse:

```
1...miner..noeud1..address...
```

Maintenant, créez un répertoire pour les clés privées RSA, utilisez openssl pour y stocker une nouvelle clé, et vérifiez:

```
mkdir␣~/.multichain/chain1/stream−privkeys/
#creation␣d'un␣fichier␣avec␣l'extension␣.pem
openssl␣genpkey␣−algorithm␣RSA␣−out␣~/.multichain/chain1/stream−privkeys/1...miner..
    noeud1..address....pem
cat␣~/.multichain/chain1/stream−privkeys/1....pem
```

Vous devriez voir un bloc de donnée alphanumérique, entouré par les lignes `BEGIN PRIVATE KEY` et `END PRIVATE KEY`. Il s'agit d'une clé privée RSA. Maintenant, générerons une clé publique à partir de cela, convertissez-le immédiatement en hexadécimal, stockez le dans la variable shell `pubkeyhex` et vérifiez:

```
pubkeyhex=$(openssl␣rsa␣−pubout␣−in␣~/.multichain/chain1/stream−privkeys/1...miner..
    noeud1..address....pem␣|␣xxd␣−p␣−c␣9999)
echo␣$pubkeyhex
```

Enfin, publions notre clé publique comme élément non étiqueté dans le flux de `pubkeys`, cette commande devrait renvoyer un ID de transaction:

```
multichain−cli␣chain1␣publishfrom␣1...miner..admin..noeud1...␣pubkeys␣''␣$pubkeyhex
```

8.4 Stockage de données chiffrées

Passons maintenant au deuxième noeud. Pour commencer, choisissons une adresse que nous utiliserons pour publier des éléments de flux confidentiels:

```
multichain−cli␣chain1␣getaddresses
```

Copier l'adresse :

```
1...miner..slave01..address...
```

Utilisez maintenant le premier noeud pour donner des autorisations globales d'envoi et de réception (pour le changement) à cette adresse:

```
multichain−cli␣chain1␣grant␣1...miner..noeud2..address...␣send,receive
```

Retour sur le deuxième noeud, créez un mot de passe aléatoire, utilisez-le pour chiffrer le fichier `/proc/cpuinfo` avec l'algorithme AES, convertissez immédiatement le résultat en hexadécimal, stockez-le dans la variable shell `cipherhex` et vérifiez :

```
password=$(openssl␣rand␣-base64␣48)
cipherhex=$(openssl␣enc␣-aes-256-cbc␣-in␣/proc/cpuinfo␣-pass␣pass:$password␣|␣xxd
    ␣-p␣-c␣99999)
echo␣$cipherhex
```

Maintenant, nous pouvons publier ce gros morceau de données hexadécimales dans le flux d'éléments (items):

```
multichain-cli␣chain1␣publishfrom␣1...miner..noeud2..address...␣items␣secret-cpuinfo␣
    $cipherhex
```

Copiez le `txid` obtenu:

```
5433...
```

Maintenant, créez un répertoire pour stocker les mots de passe de chiffrement des éléments et stockez celui-ci pour utilisation future:

```
mkdir␣~/.multichain/chain1/stream-passwords/
echo␣$password␣>␣~/.multichain/chain1/stream-passwords/5433....txt
```

8.5 Publications des mots de passe chiffrés

En restant sur le deuxième noeud, nous devons d'abord nous abonner au flux `pubkeys`, puis récupérer la clé publique RSA du premier noeud à partir de ce flux:

```
multichain-cli␣chain1␣subscribe␣pubkeys
multichain-cli␣chain1␣liststreampublisheritems␣pubkeys␣1...miner..noeud1..address...␣
    true␣1
```

Copiez le `txid` obtenu:

```
d5b1...
```

Copiez le `vout` obtenu (normalement 0):

```
VOUT_ID
```

Bien que la donnée (la clé publique) dont nous avons besoin puissent déjà être affichées dans la réponse, nous la récupérons dans une étape distincte pour faciliter la conversion de l'hexadécimal et la placer dans un fichier temporaire, dont nous pouvons visualiser le contenu:

```
multichain-cli␣chain1␣gettxoutdata␣d5b1...␣VOUT_ID␣|␣tail␣-n␣1␣|␣xxd␣-p␣-r␣>␣/
    tmp/pubkey.pem
cat␣/tmp/pubkey.pem
```

Vous devriez voir un bloc de données alphanumérique, entouré par les lignes `BEGIN PUBLIC KEY` et `END PUBLIC KEY`. Il s'agit de la clé publique RSA qui a été publiée dans le flux de `pubkeys` plus tôt. Maintenant, utilisez cette clé publique pour chiffrer le mot de passe chiffrant l'élément (le fichier `/proc/cpuinfo`), convertissez-le immédiatement en hexadécimal, stocker-le dans la variable shell `keycipherhex` et vérifiez:

```
keycipherhex=$(echo␣$password␣|␣openssl␣rsautl␣-encrypt␣-inkey␣/tmp/pubkey.pem␣-
    pubin␣|␣xxd␣-p␣-c␣9999)
echo␣$keycipherhex
```

Maintenant, publions ce mot de passe chiffré sur le flux d'accès (`access`), en utilisant le `txid` du précédent élément comme étiquette (`label`) (en référence à l'élément correspondant dans le flux `item`). Ceci permettra à l'autre partie de le retrouver facilement:

```
label=d5b1.....
multichain-cli␣chain1␣publishfrom␣1...miner..noeud2..address...␣access␣$label␣
    $keycipherhex
```

8.6 Récupération de l'élément chiffré

Revenons au premier noeud. Nous devons nous abonner aux flux `items` et aux flux `access`, puis nous nous récupérons le dernier article publié:

```
multichain-cli␣chain1␣subscribe␣'["items","access"]'
multichain-cli␣chain1␣liststreamitems␣items␣true␣1
```

Le txid montré devrait correspondre a celui de la partie précédente: `a1b2...` Copiez le `vout` obtenu (normalement 0):

```
VOUT_ID
```

Bien que la données dont nous avons besoin soient déjà disponibles dans la réponse, nous la récupérons via une étape distincte afin de faciliter son affectation à une variable shell:

```
cipherhex=$(multichain-cli␣chain1␣gettxoutdata␣5433...␣VOUT_ID␣|␣tail␣-n␣1)
```

Maintenant, récupérons le mot de passe correspondant à cet élément chiffré, qui a été crypté spécialement pour ce noeud en utilisant sa clé publique:

```
label=d5b1.....
multichain-cli␣chain1␣liststreamkeyitems␣access␣$label␣true
```

Copiez le `txid` obtenu:

```
6927...
```

Copiez le `vout` obtenu (normalement 0):

```
VOUT_ID
```

Comme précédemment, même si la donnée dont nous avons besoin peuvent déjà être disponible dans la réponse, nous la récupérerons via une étape distincte:

```
keycipherhex=$(multichain-cli␣chain1␣gettxoutdata␣6927...␣VOUT_ID␣␣|␣tail␣-n␣1)
echo␣$keycipherhex
```

Nous pouvons récupérer le mot de passe original en décryptant celui-ci à l'aide de la clé privée RSA stockée précédemment sur ce noeud:

```
password=$(echo␣$keycipherhex␣|␣xxd␣-p␣-r␣|␣openssl␣rsautl␣-decrypt␣-inkey␣~/.
    multichain/chain1/stream-privkeys/1....pem)
echo␣$password
```

Enfin, nous pouvons utiliser ce mot de passe pour déchiffrer le contenu de l'élément du flux `items`, pour révéler les données secrètes stockées (le fichier `/proc/cpuinfo` du second noeud):

```
echo␣$cipherhex␣|␣xxd␣-p␣-r␣|␣openssl␣enc␣-d␣-aes-256-cbc␣-pass␣pass:$password
```

8.7 Conclusion

Voila! Le deuxième noeud a pu envoyer son fichier `/proc/cpuinfo` au premier noeud sur la blockchain, sans que le contenu soit visible par aucun autre noeud. Le fichier pourrait être révélé à un participant supplémentaire en publiant le mot de passe dans le flux d'accès, crypté à l'aide de la clé publique RSA que le participant supplémentaire a publié dans le flux de pub. Il n'est jamais nécessaire de stocker de nouveau le grand fichier dans le flux d'éléments. Nonobstant la visibilité restreinte, tous les noeuds du réseau participent au stockage, à l'horodatage et à la notarisation des informations publiées.

Une application réelle utilisant cette technique serait peu susceptible d'utiliser les outils de ligne de commande comme dans ce tutoriel. Au lieu de cela, il aurait accès aux commandes MultiChain à l'aide de l'API JSON-RPC et utiliserait des bibliothèques de cryptage natif du langage de programmation utilisé. Les algorithmes de cryptage RSA et AES ci-dessus peuvent être remplacés par d'autres algorithmes de chiffrement symétriques et asymétrique.

9 Exercice (A rendre ?)

Créer une blockchain afin de publier les données de sondes (temperature et humidité). Les données seront publiées depuis des adresses SENSOR (chaque noeud publiera des couples T/H) et devront être chiffrées. Les adresses SENSOR n?auront le droit de publier des données que si ils ont été approuvés par les adresses ADMIN. Seules les adresses VISU auront le droits de consulter les données.

A rendre sur CELENE :

- un schéma de votre architecture (vous pouvez envoyer une photo d?un schéma)
- un mini-howto avec les commandes pour mettre en place cette architecture
- un résumé décrivant votre architecture et justifiant vos choix