Documentation - Solveur de Puzzle Bitcoin v2.1 - OPTIMISÉ

Table des matières

Vue d'ensemble

Ce script Rust est un solveur de puzzle Bitcoin haute performance conçu pour rechercher des clés privées correspondant à des adresses Bitcoin spécifiques. Il utilise une approche multi-threadée avec support CPU et GPU (simulé) pour optimiser les performances de recherche.

Fonctionnalités principales

- **Multi-threading avancé** : Support CPU, GPU et hybride
- **Algorithmes de recherche multiples** : séquentiel, aléatoire, intelligent
- **Optimisations cryptographiques** : Baby-step Giant-step, Smart Jump
- **Notifications Telegram** : Alertes automatiques lors de découvertes
- **Points de contrôle** : Sauvegarde automatique du progrès
- **Détection GPU automatique** : CUDA et OpenCL

Installation et dépendances

Prérequis système

```bash
# Ubuntu/Debian
sudo apt update
sudo apt install build-essential pkg-config libssl-dev

# CentOS/RHEL
sudo yum groupinstall "Development Tools"
sudo yum install openssl-devel

# macOS
brew install openssl
```

Installation de Rust

```bash
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.cargo/env
```

Dépendances Cargo

Ajoutez dans votre `Cargo.toml` :

```toml
[dependencies]
bitcoin = "0.31"
ibig = "0.3"
rand = "0.8"
num_cpus = "1.16"
reqwest = { version = "0.11", features = ["blocking"] }
chrono = { version = "0.4", features = ["serde"] }
```

Compilation

```bash
cargo build --release
```

Configuration

Fichier config.txt

Au premier lancement, le programme génère automatiquement un fichier `config.txt` avec les paramètres par défaut :

ini

```ini
# Fichier de configuration pour le solveur de puzzle Bitcoin OPTIMISÉ v2.1

# Plage de recherche (décimal ou hexadécimal avec préfixe 0x)
start=0x20000000000000000
end=0x3fffffffffffffff

# Nombre de coeurs CPU à utiliser (0 = détection automatique)
cores=0

# Mode de recherche : 'random', 'sequential', 'smart', 'kangaroo'
mode=smart

# Mode de calcul : 'cpu', 'gpu', 'hybrid' (cpu+gpu)
compute_mode=hybrid

# Configuration GPU
gpu_device_id=0
gpu_batch_size=50000

# Ratio CPU/GPU en mode hybride (0.5 = 50% CPU, 50% GPU)
cpu_gpu_ratio=0.5

# Après combien d'essais sauter vers un nouvel emplacement aléatoire
switch_interval=1000000

# Ratio de la taille du sous-intervalle (ex: 0.001 pour 0.1%)
subinterval_ratio=0.001

# Arrêter le programme dès qu'une clé est trouvée ? (true ou false)
stop_on_find=false

# Fichier contenant la liste des adresses Bitcoin à trouver
puzzle_file=puzzle.txt

# Algorithmes avancés
baby_steps=true
giant_steps=true
bloom_filter=false
smart_jump=true

# Paramètres de performance
batch_size=10000
checkpoint_interval=10000000

# Configuration Telegram (optionnel)
telegram_bot_token=YOUR_BOT_TOKEN_HERE
telegram_chat_id=YOUR_CHAT_ID_HERE
```

Paramètres détaillés

Paramètre	Type	Description	Exemple
`start`	String	Clé de début (hex/dec)	`0x1` ou `1`
`end`	String	Clé de fin (hex/dec)	`0xFFFFF` ou `1048575`
`cores`	usize	Nombre de threads CPU	`8` (0 = auto)
`mode`	String	Mode de recherche	`smart`, `random`, `sequential`
`compute_mode`	String	Type de calcul	`cpu`, `gpu`, `hybrid`
`gpu_batch_size`	usize	Taille des lots GPU	`50000`
`cpu_gpu_ratio`	f64	Ratio CPU/GPU en hybride	`0.5` (50/50)
`switch_interval`	u64	Intervalle de saut	`1000000`
`stop_on_find`	bool	Arrêt à la découverte	`true`/`false`

Modes de fonctionnement

1. Mode Sequential

Recherche linéaire dans la plage spécifiée.

```ini
mode=sequential
```

Avantages : Couverture complète, reproductible **Inconvénients** : Lent pour grandes plages

2. Mode Random

Recherche aléatoire dans la plage.

```ini
mode=random
switch_interval=100000
```

Avantages : Probabilité uniforme **Inconvénients** : Possible redondance

3. Mode Smart

Recherche intelligente avec patterns mathématiques.

```ini
mode=smart
smart_jump=true
```

Avantages : Optimisé pour certains types de clés **Inconvénients** : Peut manquer certaines clés

4. Modes de calcul

CPU uniquement

```ini
compute_mode=cpu
cores=8
```

GPU uniquement (simulé)

```ini
compute_mode=gpu
gpu_device_id=0
gpu_batch_size=50000
```

Hybride CPU+GPU

```ini
compute_mode=hybrid
cpu_gpu_ratio=0.7
```

Exemples d'utilisation

Exemple 1 : Recherche simple puzzle #64

```ini
# Configuration pour puzzle Bitcoin #64
start=0x8000000000000000
end=0xFFFFFFFFFFFFFFFF
mode=random
compute_mode=cpu
cores=4
puzzle_file=puzzle64.txt
stop_on_find=true
```

Fichier `puzzle64.txt` :

```
16jY7qLJnxb7CHZyqBP8qca9d51gAjyXQN
```

Exemple 2 : Recherche haute performance avec GPU

ini

```ini
# Configuration optimisée pour GPU
start=0x20000000000000000
end=0x3fffffffffffffff
mode=smart
compute_mode=hybrid
cpu_gpu_ratio=0.3
gpu_batch_size=100000
batch_size=50000
smart_jump=true
checkpoint_interval=5000000
```

Exemple 3 : Recherche avec notifications Telegram

ini

```ini
# Configuration avec alertes
mode=random
compute_mode=cpu
cores=0
telegram_bot_token=123456789:ABCdefGHIjklMNOpqrsTUVwxyz
telegram_chat_id=-100123456789
stop_on_find=true
```

Exemple 4 : Recherche de plage personnalisée

ini

```ini
# Recherche dans une plage spécifique
start=1000000000000000000
end=9999999999999999999
mode=sequential
compute_mode=cpu
cores=16
checkpoint_interval=1000000
```

Architecture technique

Structure des threads

```
Main Thread
├── CPU Workers (0..n)
│   ├── Worker 0: Range [start, start+chunk]
│   ├── Worker 1: Range [start+chunk, start+2*chunk]
│   └── ...
├── GPU Workers (0..m)
│   ├── GPU Worker 0: Batch processing
│   └── GPU Worker 1: Batch processing
└── Statistics Thread
```

Algorithmes implémentés

1. Génération de patterns de clés

```rust
fn generate_key_patterns(base_key: &UBig, rng: &mut FastRng) -> Vec<UBig>
```

- Inversion des chiffres
- Addition/soustraction de nombres de Fibonacci
- Multiplication par facteurs premiers
- Permutation des chiffres

2. Génération d'adresses

```rust
fn generate_address_variants(secp: &Secp256k1<All>, secret_key: &SecretKey) -> Vec<(PrivateKey,
```

- Adresses compressées
- Adresses non compressées
- Format P2PKH

3. Points de contrôle

```rust
fn save_checkpoint(current_key: &UBig, core_id: usize)
fn load_checkpoint(core_id: usize, default_start: &UBig) -> UBig
```

Optimisations et performances

Optimisations CPU

- **FastRng** : Générateur pseudo-aléatoire optimisé
- **Pattern recognition** : Génération intelligente de clés candidates
- **Batch processing** : Traitement par lots pour réduire les appels système

Optimisations mémoire

- **UBig** : Arithmétique sur grands entiers efficace
- **HashSet** : Recherche O(1) pour les adresses
- **Arc/Mutex** : Partage de données thread-safe minimal

Benchmarks typiques

Configuration	Performance approximative
CPU 8 cores	50K-200K clés/seconde
GPU simulé	500K-2M clés/seconde
Hybride	800K-3M clés/seconde

Conseils d'optimisation

1. **Ajustez la taille des lots**

```ini
batch_size=10000        # Pour CPU
gpu_batch_size=50000    # Pour GPU
```

2. **Optimisez le ratio CPU/GPU**

```ini
cpu_gpu_ratio=0.3  # 30% CPU, 70% GPU si GPU puissant
```

3. **Configurez les points de contrôle**

```ini
checkpoint_interval=10000000  # Sauvegarde tous les 10M de clés
```

Fichiers de sortie

found.txt

Fichier principal contenant les clés trouvées :

```
[2024-06-24 20:15:30] [CPU 2] Trouvé! Clé (hex): 1a2b3c4d5e6f7890, Adresse:
1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2
[2024-06-24 20:16:45] [GPU] Trouvé! Clé privée (hex): 9f8e7d6c5b4a3210, Adresse:
3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy
```

checkpoint_core_X.txt

Points de contrôle par thread :

```
12345678901234567890
```

Fichier puzzle.txt

Liste des adresses à rechercher :

```
1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2
3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy
bc1qxy2kgdygjrsqtzq2n0yrf2493p83kkfjhx0wlh
```

Dépannage

Erreurs courantes

1. "Clé de départ invalide"

```
bash
```

```
Erreur: Clé de départ invalide.
```

Solution : Vérifiez le format des clés dans config.txt

```
ini
```

```ini
# Correct
start=0x1000000000000000
# ou
start=1152921504606846976

# Incorrect
start=0xGGGGGGGGGGGGGGGG
```

2. "GPU détecté mais non fonctionnel"

```
bash
```

```
⚠️ Aucun GPU détecté. Mode CPU configuré par défaut.
```

Solution : Vérifiez les drivers GPU

```bash
# NVIDIA
nvidia-smi

# AMD
rocm-smi

# Passer en mode CPU
compute_mode=cpu
```

3. "Fichier puzzle vide"

```bash
Erreur: Le fichier puzzle 'puzzle.txt' est vide ou n'a pas pu être lu.
```

Solution : Créez le fichier puzzle.txt avec les adresses à rechercher

4. Performance faible

Diagnostic : Vérifiez la sortie des statistiques

```
[Temps: 00:05:30] [Total: 0.05 Mk/s] [CPU: 0.05 Mk/s | GPU: 0.00 Mk/s] [Trouvées: 0]
```

Solutions :

- Augmentez `batch_size`
- Réduisez `checkpoint_interval`
- Utilisez le mode `random` au lieu de `sequential`
- Activez `smart_jump=true`

Logs de débogage

Pour activer les logs détaillés :

```bash
RUST_LOG=debug ./bitcoin_puzzle_solver
```

Monitoring des ressources

bash

```bash
# Surveillance CPU
htop

# Surveillance GPU
nvidia-smi -l 1

# Surveillance mémoire
free -h
```

Sécurité et considérations légales

⚠️ AVERTISSEMENTS IMPORTANTS

1. **Usage légal uniquement**
   - Ce script est destiné à des fins éducatives et de recherche
   - N'utilisez que sur vos propres clés ou avec autorisation explicite
   - Respectez les lois locales sur la cryptographie
2. **Sécurité des clés privées**
   - Les clés trouvées sont stockées en clair dans `found.txt`
   - Chiffrez ce fichier ou supprimez-le après usage
   - Ne partagez jamais les clés privées découvertes
3. **Consommation de ressources**
   - Le programme utilise intensivement CPU/GPU
   - Surveillez la température du système
   - Utilisez avec précaution sur ordinateurs portables

Bonnes pratiques

1. **Backup des configurations**

bash

```bash
cp config.txt config_backup_$(date +%Y%m%d).txt
```

2. **Rotation des logs**

bash

```bash
# Archiver les anciens résultats
tar -czf found_$(date +%Y%m%d).tar.gz found.txt checkpoint_*.txt
```

3. **Test sur petites plages**

```ini
# Test initial
start=1
end=1000000
mode=sequential
```

Configuration de sécurité Telegram

1. **Créer un bot** :
   - Parlez à @BotFather sur Telegram
   - Utilisez `/newbot` et suivez les instructions
   - Notez le token fourni
2. **Obtenir le chat_id** :
   - Ajoutez @userinfobot à votre chat
   - Le bot vous donnera votre chat_id
3. **Tester la configuration** :

```bash
curl -X POST "https://api.telegram.org/bot<TOKEN>/sendMessage" \
     -d "chat_id=<CHAT_ID>&text=Test de notification"
```

Conclusion

Ce solveur de puzzle Bitcoin v2.1 offre une solution complète et optimisée pour la recherche de clés privées. Avec ses multiples modes de fonctionnement, ses optimisations avancées et son architecture multi-threadée, il constitue un outil puissant pour les chercheurs et les passionnés de cryptographie.

Rappel important : Utilisez cet outil de manière responsable et conformément aux lois en vigueur dans votre juridiction.

Documentation générée pour le Solveur de Puzzle Bitcoin v2.1 - Juin 2024