Electrum Documentation

リリース *3.3*

Thomas Voegtlin

2019 年 07 月 31 日

目次

Electrum is a lightweight Bitcoin wallet.

第 1 章

GUI and beginners

1.1 よくある質問

1.1.1 Electrum はどのように動作しますか？

Electrum が焦点にあてているのはスピード、少ない計算資源の使用量、Monacoin を簡単にすることです。Monacoin のシステムの最も複雑な部分は高性能なサーバーが操作し、Electrum はこれと連携して動作するので起動時間はわずかです。

1.1.2 Electrum はサーバーを信頼していますか？

そうでもないです;Electrum クライアントは秘密鍵をサーバーに送信しません。さらに、サーバーから受け取った情報は Simple Payment Verification=SPV と呼ばれる技術で検証されます。

デフォルトでは、Electrum は最大 10 台のサーバへの接続を維持しようとします。クライアントは、これらのサーバーに対してブロックヘッダ通知をサブスクライブします。接続されているサーバーのうち、1 台を除くすべてのサーバーで使用されます。複数のソースからブロックヘッダーを取得すると、遅延サーバー、チェーン分割およびフォークの検出に役立ちます。

いずれかのサーバーが「メイン」サーバーとして任意に選択されます。

- クライアントは、自身のアドレス（詳細：scriptPubKeys の sha256 ハッシュ）をサブスクライブし、そのアドレスへの新しいトランザクションが通知されるようにします。また、アドレスの既存の履歴も同期されます。これはクライアントがサーバーのプライバシーを犠牲にすることを意味します。何故ならサーバ側はこれらのアドレスがすべて同じ相手であると合理的に推測できるからです。
- 上記のように、確認されたトランザクションは SPV によって検証されます。
- サーバーは未確認のトランザクションについて一時的に信頼しています。

- サーバーは手落ちにより嘘をつくことができます。つまりクライアントの (確認済み、未確認の両方の) トランザクションについて忘れることができるということです。
- メインサーバーは手数料計算にもまた使われ、それをクライアントは信用します。(クライアントには様々なレイヤの健全性チェックが適用されます)
- メインサーバーはクライアントが作成したトランザクションをブロードキャストすることにも使われます。
- サーバーのピアリストはクライアントによって要求され、使用できる他のサーバーを記憶します。(初期動作のための直接ソースコードに書き込まれたサーバーのリストがクライアントにあります。)

更に、接続されている全てのサーバーがクライアントの IP アドレス（これはプロキシ、VPN、Tor のアドレスの場合もあります）を認識します。

上記のようなプライバシー損失を犠牲にして、迅速な始動時間および低いリソース使用が達成されます。プロトコルとクライアントは、サーバーの信頼性を最小限に抑えるように設計されています。

誰でもサーバーを動かすことは可能です。もしプライバシーに強い関心があり、SPV によるセキュリティ保証だけでは不十分な場合は独自の Electrum サーバーの運用を検討してください。

1.1.3 シードとは何ですか？

シードは秘密鍵を生成するために使用されるランダムなフレーズです。

例：

```
slim sugar lizard predict state cute awkward asset inform blood civil sugar
```

あなたのウォレットはシードから完全に復元することができます。そのためには、インストールウィザードで「I already have a seed（既存のシードを使用する)」オプションを選択してください。

1.1.4 シードはどれくらい安全ですか？

Electrum によって作成されたシードフレーズは 132bit のエントロピーを持ちます。つまり、Monacoin の秘密鍵（長さ 256bit）と同じレベルのセキュリティを提供します。実際、長さ n の楕円曲線キーは、n / 2bit のセキュリティを提供します。

1.1.5 パスワードを忘れてしまいました。何ができるでしょう。

パスワードを復元することはできません。ただし、シードフレーズからウォレットを復元し、新しいパスワードを選ぶことができます。パスワードとシードの両方がわからなくなった場合、あなたの資金を取り戻す方法はありません。これがシードフレーズを紙に書き留めるようにお願いする理由です。

シードフレーズからウォレットを復元するには、create a new wallet を選んだのち、「I already have a seed」を選択してシードフレーズを入力してください。

1.1.6 私のトランザクションが長い間承認されていません。何ができますか？

Monacoin トランザクションはマイナーがブロックチェーンに対してその書き込みを許可した時に「承認」されます。一般に承認スピードはあなたがトランザクションに添付した手数料に依存します。マイナーは最も高い手数料を支払うトランザクションを優先します。

Electrum の最近のバージョンでは、トランザクションに支払う手数料を十分にするために「ダイナミックフィー」を使用しています。この機能は Electrum の最近のバージョンではあらかじめ有効になっています。

未承認のトランザクションを作成してしまった場合、次の操作を実行できます。：

- しばらく待つ。最終的にはあなたのトランザクションは承認されるかキャンセルされます。これには数日かかることがあります。
- トランザクション手数料を増やす。これは「置き換え可能な (replaceable)」トランザクションでのみ可能です。このタイプのトランザクションを作成するには、トランザクションを送信する前に、[送信 (send)] タブで [Replaceable] をチェックしておく必要があります。[send] タブの [Replaceable] オプションが表示されない場合は、[ツール (Tool)] メニュー> [設定 (Preference)] > [手数料 (Fee)] タブに移動し、[Propose Replace-By-Fee] を [Always] に設定します。置き換え可能なトランザクションの場合、history タブの日付列に「Replaceable」と表示されます。交換可能な取引の手数料を増額するには、[履歴 (history)] タブのエントリを右クリックし、「手数料を増やす (Increase Fee)」を選択します。適切な料金を設定し、「OK」をクリックします。未署名のトランザクションがウィンドウにポップアップ表示されます。「署名 (Sign)」をクリックして「発信 (Broadcast)」をクリックします。
- 「親のための子どもの支払い (Child Pays for Parent)」トランザクションの作成をする。CPFP はその親であるトランザクションのわずかな手数料を補うために高い手数料を支払おうとする新しいトランザクションです。これは資金の受領者によってのみ、またはトランザクションがお釣りアウトプットを場合に送信者が行うことができます。CPFP トランザクションを作成するには、[履歴 (history)] タブの未承認のトランザクションを右クリックし [Child pays for parent] を選択します。適切な手数料を設定したら [OK] をクリックします。未署名のトランザクションがウィンドウにポップアップ表示されます。「署名 (Sign)」をクリックして「発信 (Broadcast)」をクリックします。

1.1.7 Electrum のアドレスを「フリーズ」するとはどういう意味ですか？

アドレスをフリーズすると、そのアドレスの資金は Monacoin の送信に使用されません。フリーズされていないアドレスに十分な資金がない場合、Monacoin は送信できません。

1.1.8 ウォレットはどのように暗号化されていますか？

Electrum は、別々の 2 つのレベルの暗号化を使用しています。

- シードと秘密鍵は AES-256-CBC を使用して暗号化されます。秘密鍵は、トランザクションに署名する必要がある短かい間だけ復号されます。このためにはあなたはパスワードを入力する必要があります。これは、保護が必要な情報がコンピュータのメモリ内で暗号化されていない時間を最小限に抑えるために行われます。
- さらに、ウォレットファイルは Wallet ファイルはディスク上で暗号化されている可能性があります。暗号化されている場合は、ウォレットを開くためにパスワードを求められます。パスワードはメモリには保持されません。Electrum は非対称暗号化（ECIES）をしているため、ウォレットをディスクに保存する際にパスワードは必要ありません。

ウォレットファイルの暗号化は、バージョン 2.8 以降ではデフォルトで有効になっています。これはあなたのプライバシーを保護することを目的としていますが、あなたが管理していないウォレットにおいて Monacoin を請求できないようにするためでもあります。

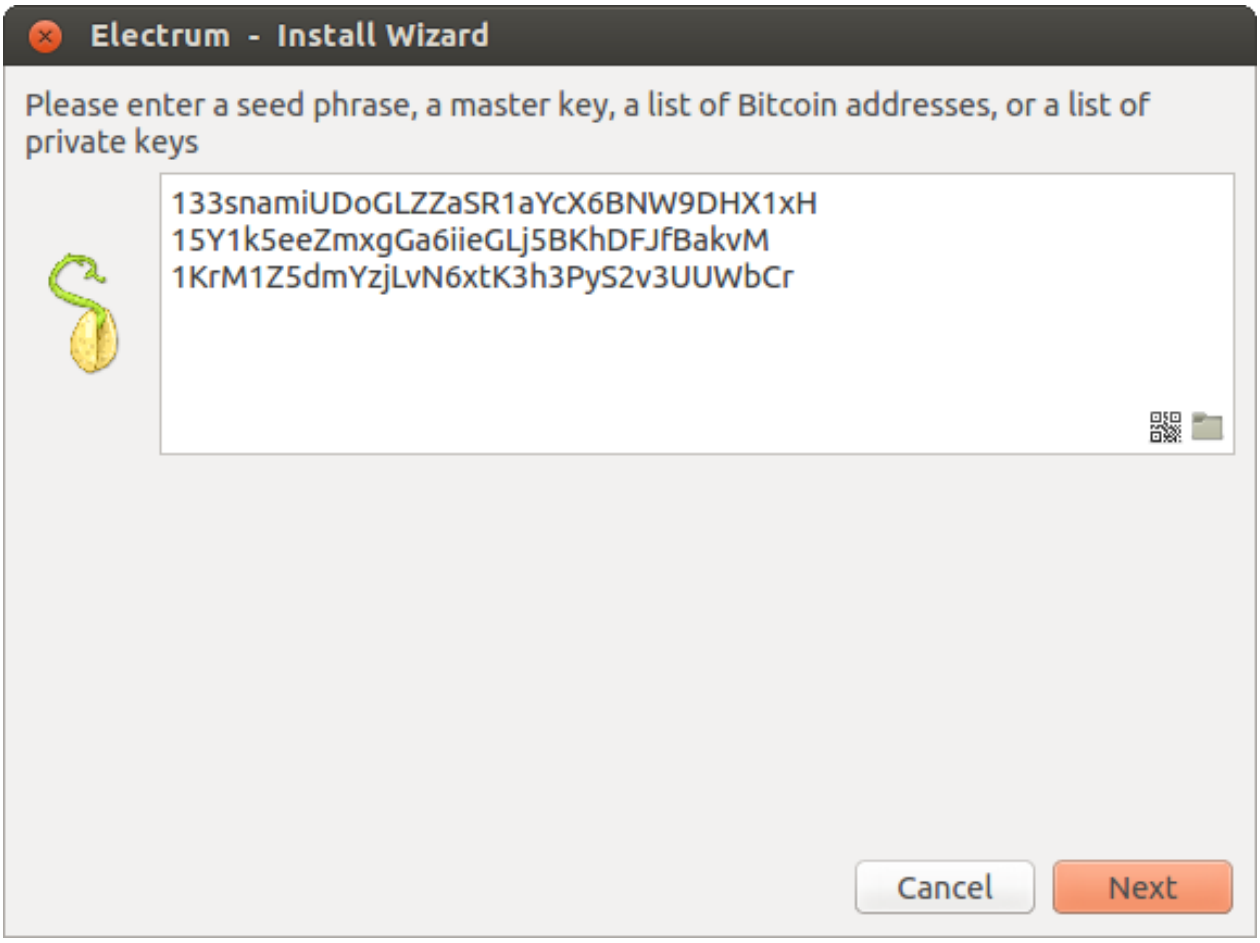
1.1.9 Electrum はコールドウォレットをサポートしていますか？

はい、ref：‘Cold Storage <coldstorage>‘を参照してください。

1.1.10 他の Monacoin クライアントから秘密鍵をインポートできますか？

Electrum 2.0 では、シードを持つウォレット内に秘密鍵をインポートすることはできません。代わりにそれらをスイープするしなくてはなりません。

秘密鍵をスイープせずにインポートしたい場合は、シードを持たない特別なウォレットを作成する必要があります。このためには、新しいウォレットを作成し「復元 (restore)」を選択し、シードを入力するか、秘密鍵のリストを入力するか、閲覧専用ウォレットを作成する場合はアドレスのリストを入力します。



このウォレットはシードから復元できないため、バックアップする必要があります。

1.1.11 他の Monacoin クライアントから秘密鍵をスイープすることはできますか？

秘密鍵のスイープとは、その秘密鍵が管理しているすべての Monacoin をあなたのウォレットの既存アドレス宛に送信することを意味します。スイープする秘密鍵はウォレットの一部にはなりません。代わりに、その秘密鍵が管理しているすべての Monacoin はあなたのウォレットのシードから確定的に生成されたアドレスに対して送信されます。

秘密鍵をスイープするには、「ウォレット (wallet)」メニュー -> 「秘密鍵 (Private Key)」 -> 「スイープ (Sweep)」に移動します。適切なフィールドに秘密鍵を入力します。「アドレス (Address)」フィールドは変更しないでください。それは宛先アドレスであり、あなたの既存の electrum ウォレットから選ばれています。「スイープ (Sweep)」をクリックします。「送信 (send)」タブに移動するので適切な手数料を設定したらコインをウォレットに送信するために「送信 (Send)」をクリックします。

1.1.12 Electrum のデータディレクトリはどこにありますか？

Electrum のデータディレクトリには、Wallet ファイル、設定ファイル、ログ、ブロックチェーンヘッダーなどが保存されます。

Windows の場合：

- 隠しファイルを表示する
- \Users\YourUserName\AppData\Roaming\Electrum（または% APPDATA %\Electrum）に移動

Mac の場合：

- Finder を開く
- フォルダに移動し（shift + cmd + G）、~/.electrum と入力

Linux の場合

- Home フォルダ
- ロケーションに移動して ~/.electrum と入力

1.1.13 ウォレットファイルはどこにありますか？

デフォルトの Wallet ファイルは default_wallet と呼ばれ、アプリケーションを最初に実行したときに作成され、/wallets フォルダに格納されています。

1.1.14 デバッグログを有効にするにはどうしたらいいですか？

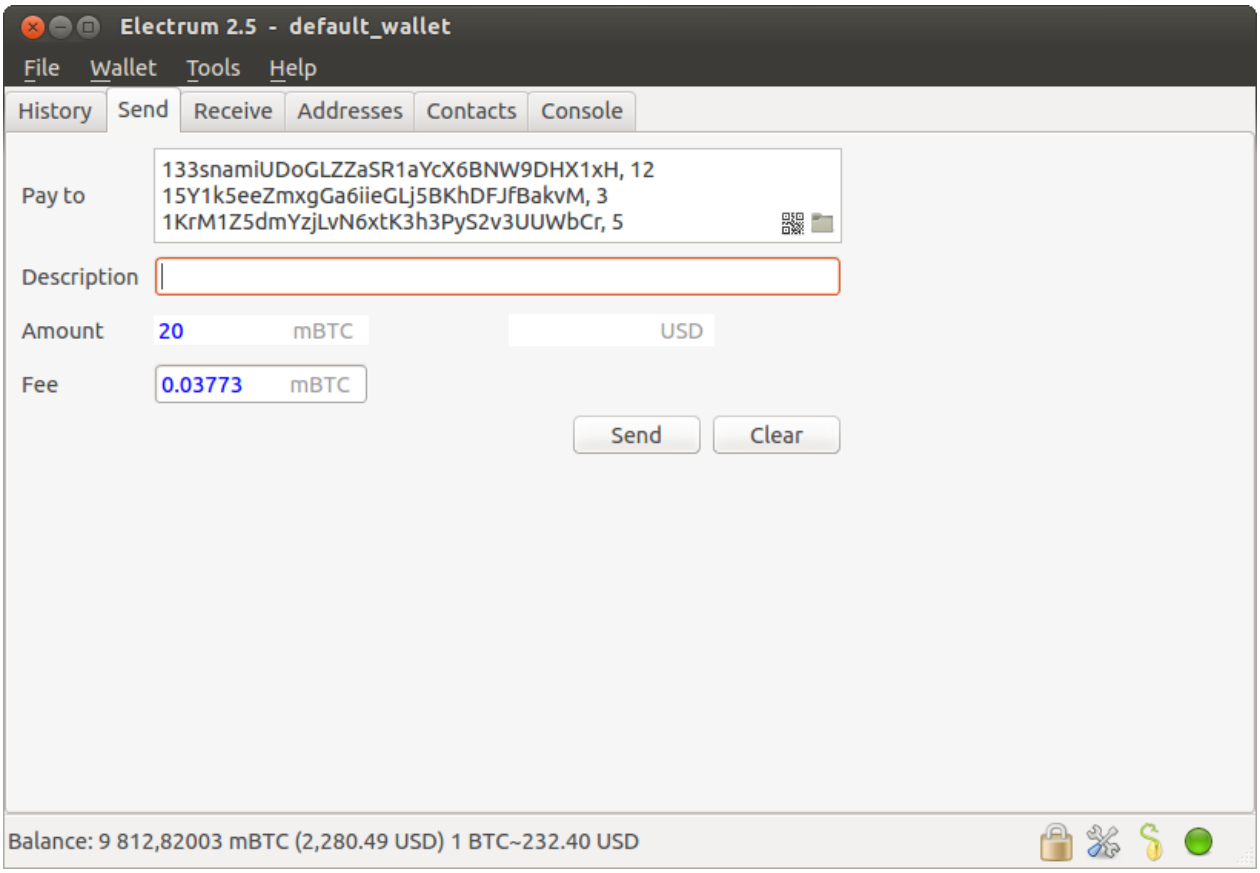
Linux/Mac では, terminal から Electrum を実行する際に -v オプションを付けることで terminal(stderr) にデバッグログが流れます。このオプションは Windows では動きません。

バージョン 3.3.5 からはディスク上にログが保存されます。これは Windows もです。

Qt GUI を使用する場合、「ツール (Tools)」メニュー -> 「設定 (Preferences)」 ->「General タブ (General Tab)」、「ログをファイルに書き込む (Write logs to file)」のチェックを入れます。Electrum の再起動後、デバッグログが /logs フォルダーに書き込まれるようになります。

1.1.15 Electrum で一括支払いができますか？

複数の出力を持つトランザクションを作成することができます。GUI では各アドレスとその送信額を 1 行に、カンマで区切ることで入力します。



金額 (Amount) は現在クライアントに設定されている単位で指定します。合計が GUI に表示されます。

また、フォルダアイコンをクリックして [支払 (Pay to)] フィールドに CSV ファイルをインポートすることもできます。

1.1.16 Electrum は生のトランザクションを作成して署名することはできますか？

Electrum では、フォームを使用してユーザーインターフェイスから生のトランザクションを作成し署名することができます。

1.1.17 Monacoin を送信しようとすると Electrum がフリーズします。

これは多数のトランザクションアウトプットを費やそうとしている場合（たとえば Monacoin の faucet から数百もの寄付を集めた場合など）に発生する可能性があります。Monacoin を送信する際に、Electrum は新しいトランザクションを作成するためにウォレット内にある未使用のコインを探します。未使用のコインは、物理的な効果や紙幣と同じように異なった数値を持つことができます。

このような場合は、ウォレットアドレスの 1 つに少量の Monacoin を送信してトランザクションインプットを統合する必要があります。これはたくさんの 5 セント硬貨のを 1 ドル紙幣と交換するのと同じです。

1.1.18 gap limit とは何ですか？

gap limit とは決定性を持つ一連のアドレスのうち連続して使用されていないアドレスの最大数です。アドレスをどこまで検索したのち停止するかを決めるために Electrum はこれを使用しています。Electrum 2.0 では、デフォルトで 20 に設定されているので、クライアントは 20 の未使用アドレスが見つかるまですべてのアドレスを取得します。

1.1.19 新しいアドレスを事前に生成するにはどうすればよいですか？

Electrum は、あなたが ‘gap limit‘に達するまで、新しいアドレスを生成してそれらを使用します。

さらに多くのアドレスを事前に生成する必要がある場合は、コンソールに wallet.create_new_address（False）と入力してアドレスを事前に生成することができます。このコマンドは新しいアドレスを 1 つ生成します。アドレスは、「アドレス (Address)」タブに赤い背景で表示され、gap limit を超えていることを表します。gap が埋まるまで赤色のままです。

警告：gap limit を超えたアドレスは自動的にはシードから回復されません。回復するには、クライアントの gap limit を増やすか、使用されたアドレスが見つかるまで新しいアドレスを生成する必要があります。

複数のアドレスを生成する場合は"for"ループを使用できます。たとえば 50 個のアドレスを生成する場合には次のようにします。

```
[wallet.create_new_address(False) for i in range(50)]
```

1.1.20 Electrum をアップグレードするには？

警告：警告：アップグレードを実行する前に、必ず紙にウォレットのシードを保存してください。

Electrum をアップグレードするには、単に最新バージョンをインストールするだけです。方法はお使いの OS によって異なります。

ウォレットファイルはソフトウェアとは別に保管されるため、OS が行わない場合には自分自身でソフトウェアの古いバージョンを安全に削除できます。

一部の Electrum アップグレードでは、ウォレットファイルの形式が変更されます。

このため、一度新しいバージョンでウォレットファイルを開いてから Electrum を古いバージョンにダウングレードすることはお勧めしません。古いバージョンでは新しいウォレットファイルを常に読み取ることができるとは限りません。

Electrum 1.x の Wallet を Electrum 2.x にアップグレードするときは、次の点を考慮する必要があります。

- Electrum 2.x では、アップグレード処理中にすべてのアドレスを再生成する必要があります。Electrum が準備完了するまで待ってください。またその際には通常より少し多く時間がかかると考えてください。

- ウォレットファイルの中身は Electrum2 ウォレットに置き換えられます。これは一度アップグレードが完了すると、Electrum 1.x はウォレットを使用できなくなることを意味します。
- 始めて Electrum2 を起動したときは「アドレス (Addresses)」タブにはアドレスは表示されません。これは想定された動作です。アップグレードが完了したら Electrum2 を再起動してください。そうすればアドレスは利用可能になります。
- Electrum のオフラインコピーには、ネットワークと同期できないためアドレスはまったく表示されません。コンソールに次のように入力すると、少数のアドレスをオフライン生成するように強制できます。：wallet.synchronize()　完了したら Electrum を再起動してください、するとあなたのアドレスが再び利用可能になります。

1.1.21 アンチウィルスソフトが Electrum をマルウェアと認識しました！

Electrum バイナリはしばしばアンチウィルスソフトによってマルウェアと認識されます。私たちにできることは何もないので、報告するのはやめてください。アンチウイルスソフトは、プログラムがマルウェアかどうかを判断するためにヒューリスティクスを使用します。これにより、誤検出が発生することがよくあります。

プロジェクトの開発者を信頼できる場合は、Electrum バイナリの GPG 署名を確認し、ウイルス対策の警告を無視しても問題ありません。

最後に、マルウェアが本当に心配な場合は、アンチウイルスソフトに依存するオペレーティングシステムを使用しないでください。

1.1.22 Electrum には最新の Python が必要ですが、私が使っている Linux ディストリビューションはまだ対応していません。どうするべきですか？

これはいくつかの方法で解決可能です。

1. 私たちが配布している AppImage を使う。これは依存している全てのものを含んだ自己解凍型バイナリです。現在、このバイナリは x86_64(amd64) アーキテクチャ用にのみ配布されています。ダウンロードして (GPG sig の確認) 実行可能にして実行するだけです。 E.g.:

```
wget https://download.electrum.org/3.3.4/electrum-3.3.4-x86_64.AppImage
chmod +x electrum-3.3.4-x86_64.AppImage
./electrum-3.3.4-x86_64.AppImage
```

2. backports を使う (e.g. Debian では stable-backports にあるパッケージをチェックしてください)
3. ディストリビューションをアップグレードする (e.g. 安定版ではなく Debian のテスト版を使う)
4. 自分で Python をコンパイルし、pip(配布用のパッケージ・マネージャーには、それによってパッケージ化されるバージョンの Python 用の PyQt5 しかないため) を使用して pyqt5 をインストールします。

```
python3 -m pip install --user pyqt5
```

(残念なことに pip の pyqt5 は x86/x86_64 にしかありません。arch 等では、 Qt/PyQt を自分でビルドする必要があるかもしれません。)

5. より新しいパッケージを持つ別の Linux ディストリビューションを実行する仮想マシンを使用します。

1.1.23 自分でサーバーを動かすかもしれません。クライアント/サーバ接続は認証されていますか。

Electrum は、エンドポイントが Electrum プロトコルを介するクライアントサーバーアーキテクチャを使用します。Electrum プロトコルは JSON-RPC ベースです。クライアントがサポートする 2 つの主なスタックは次のとおりです。

1. JSON-RPC over SSL/TLS over TCP
2. JSON-RPC over TCP

どちらのオプションも HTTP を使用しないことに注意してください。

クライアントは、SSL(平文 TCP は使われない) を介してのみサーバーに接続します。Electrum 3.1 より前は、これを切り替えるためのチェックボックスが GUI にありましたが、削除されました。

認証に関しては、クライアントは CA 署名証明書と自己署名 SSL 証明書の両方を受け入れます。サーバに最初に接続するときに、そのサーバーが CA 署名または自己署名のどちらの証明書を使用しているかを示します。

- 自己署名されている場合は、そのサーバー (TOFU) の有効期限が切れるまで、その証明書だけを受け付けます。
- もしそれが CA 署名されているなら、それは永久にそのサーバの CA 署名証明書だけを受け入れるでしょう。

サーバーを構築するうえで、CA 署名証明書と自己署名証明書の両方に利点があります。

- 自己署名証明書では、クライアントが TOFU を使用しているため、最初の接続中に man-in-the-middle が発生する可能性があります。
- CA 署名証明書を使用する場合は、証明機関を信頼する必要があります。

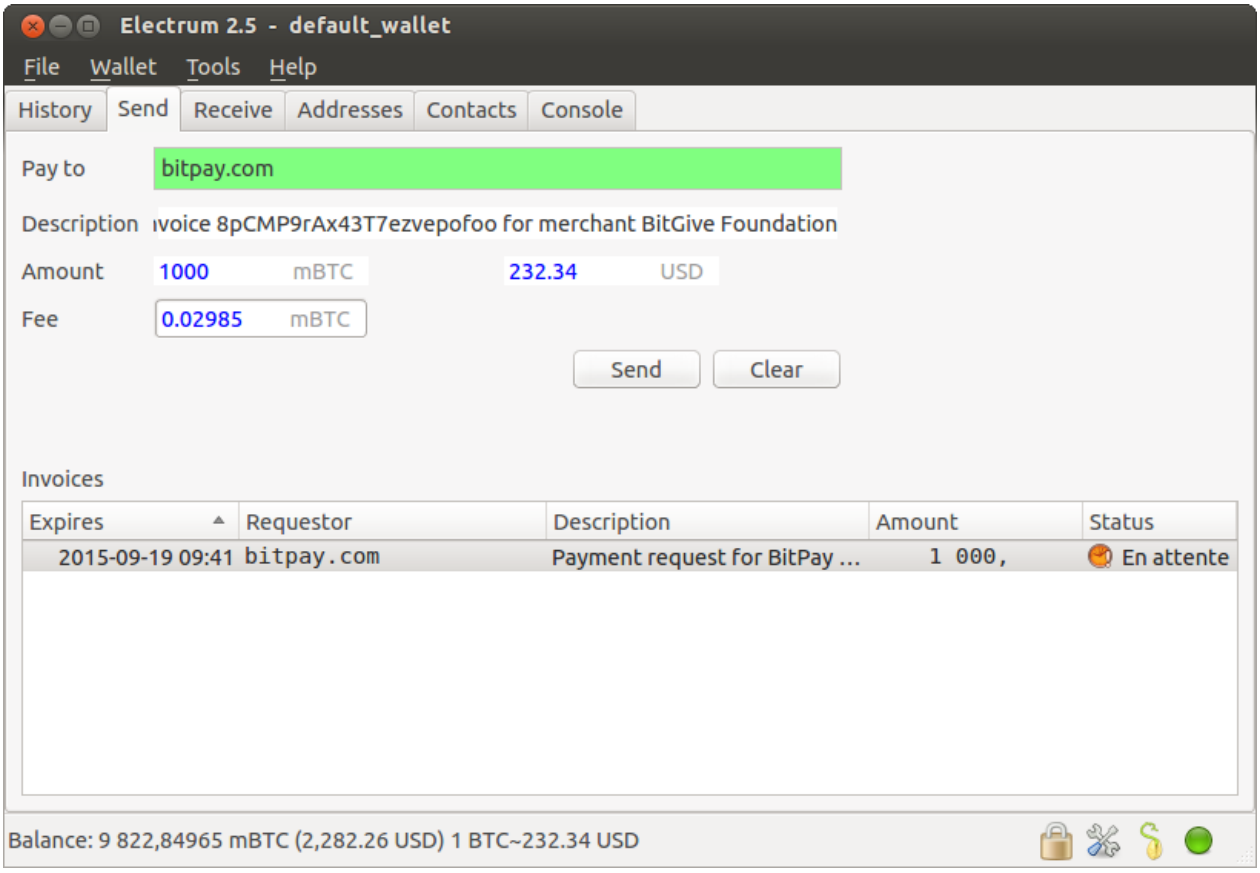
1.2 請求書-インボイス-

インボイスは請求主によって署名された支払いリクエストです。

bitcoin リンクをクリックすると URL が Electeum に渡されます。

```
electrum "bitcoin:1KLxqw4MA5NkG6YP1N4S14akDFCP1vQrKu?amount=1.0&amp;r=https%3A%2F
↪%2Fbitpay.com%2Fi%2FXxaGtEpRSqckRnhsjZwtrA"
```

これにより支払いリクエストとともに「送信 (Send)」タブが開きます。



「送金先 (Pay To)」フィールドの緑色は支払いリクエストが bitpay.com の証明書で署名され、Electrum が一連の署名を検証したことを意味します。

「送信 (Send)」タブにはインボイスのリストとそのステータスが含まれています。

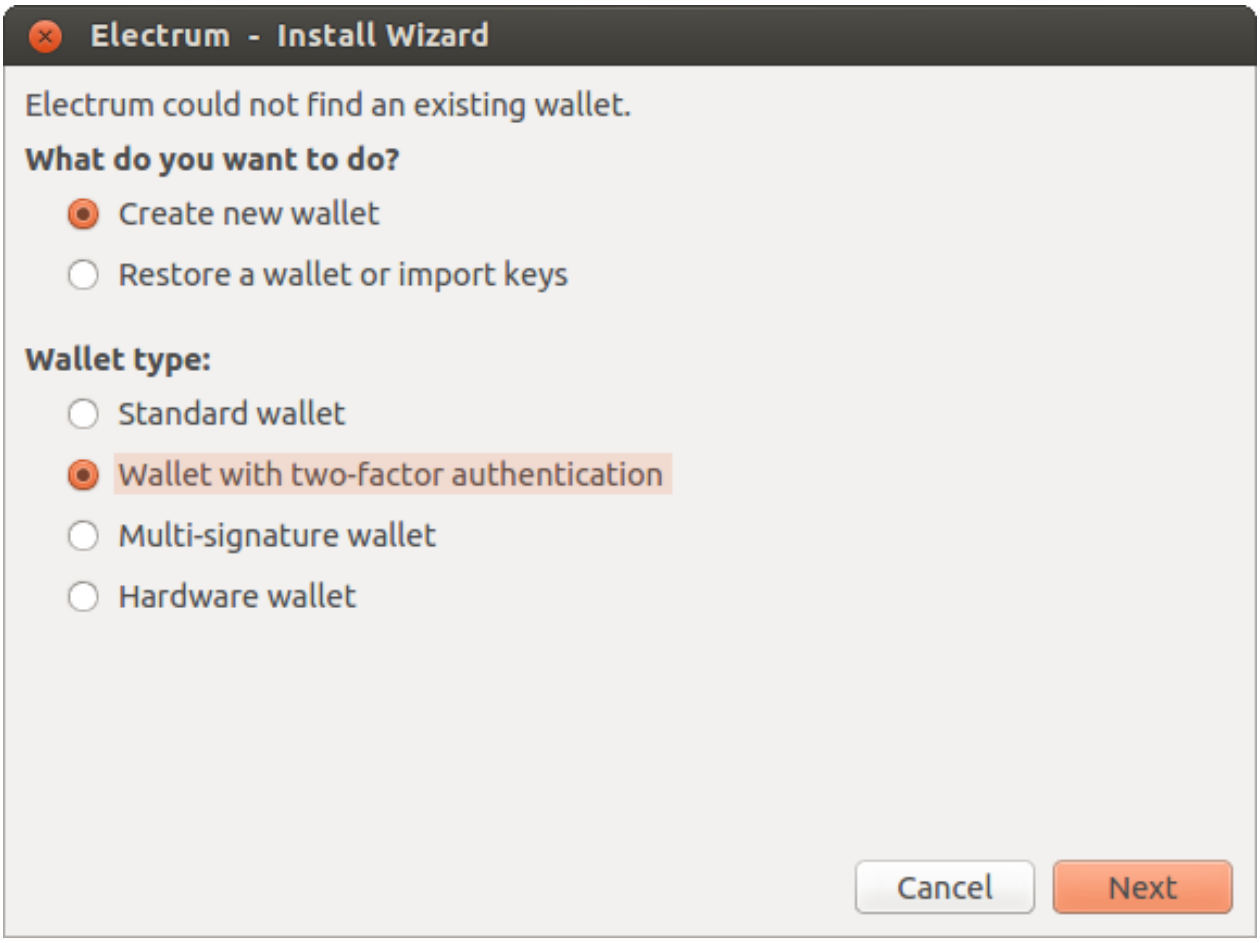
1.3 Two Factor Authentication

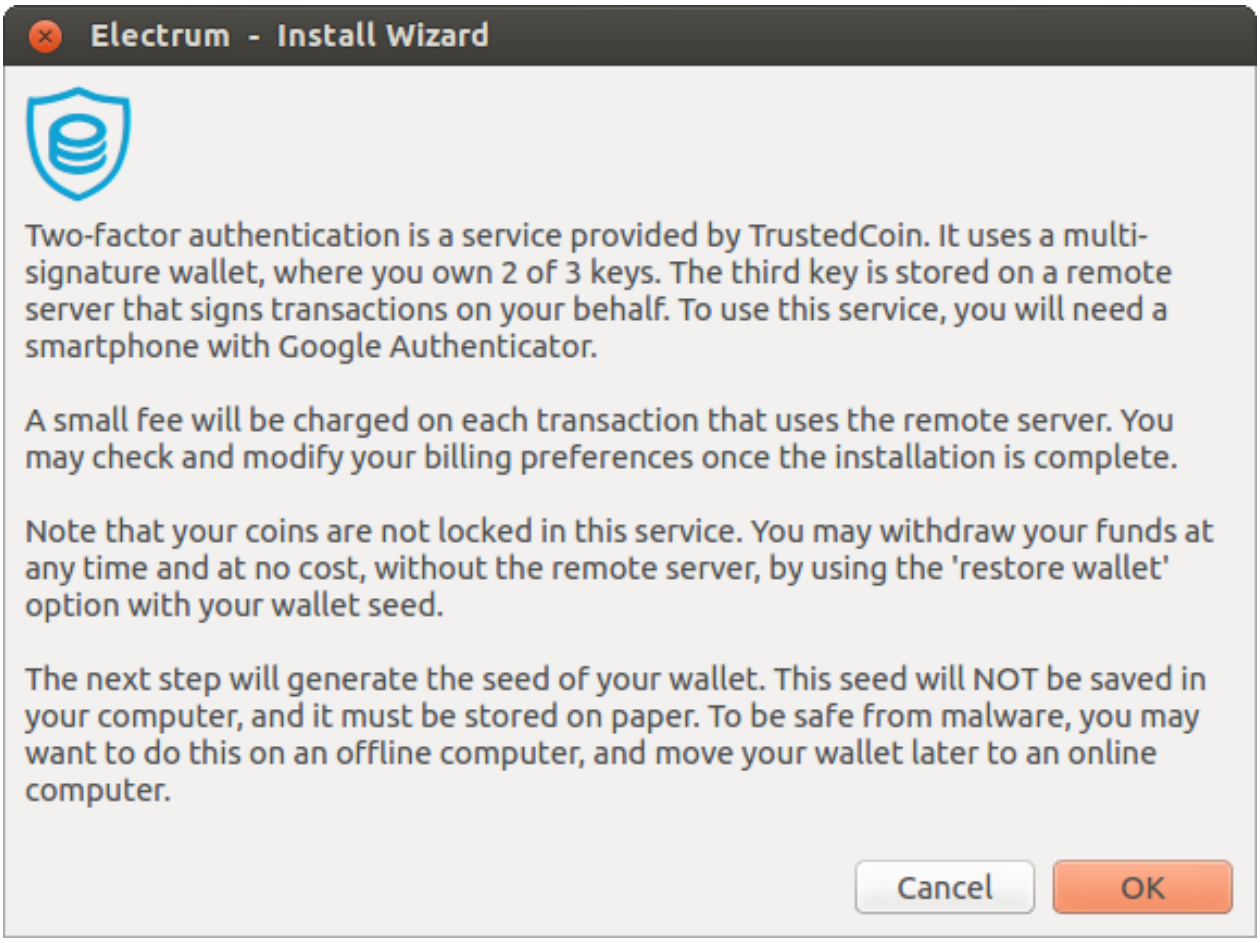
1.4 2 段階認証

Electrum はトランザクションに共同署名するように動作するリモートサーバを使い、あなたのコンピュータが感染した場合には違うレベルのセキュリティを追加する 2 段階認証ウォレットを提供しています。

問題のリモートサーバは TrustedCoin によってサービスが提供されています。それがどのように動いているかについては下のガイドがあります。

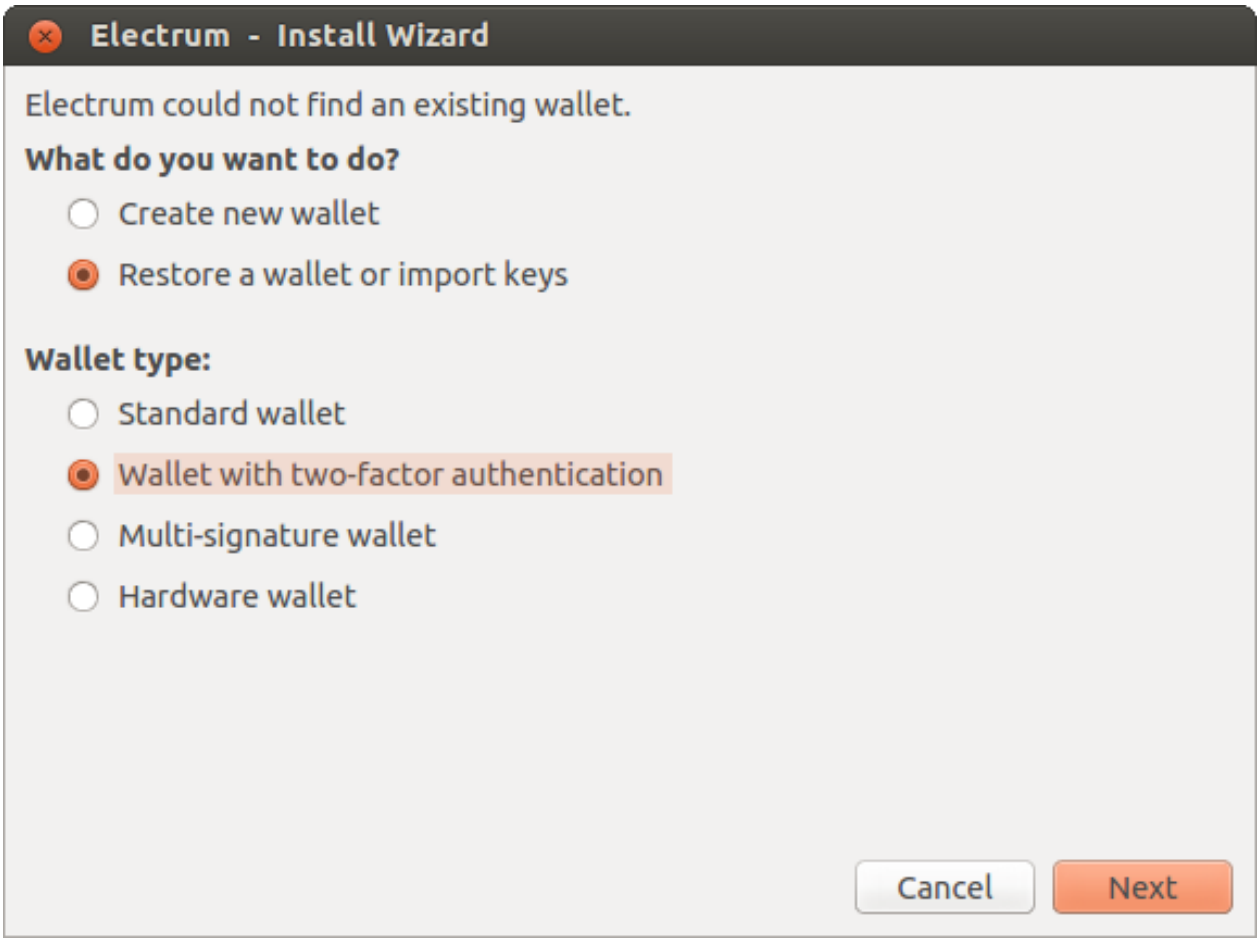
1.4.1 ウォレットを作成する





1.4.2 Seed から復元する

もし TrustedCoin が感染するかオフラインになったとしても、あなたがウォレットの Seed を持っている限りコインは安全です。あなたの Seed は 2of3 セキュリティスキームの中の二つのマスター秘密鍵を含んでいます。加えて、三つ目のマスター公開鍵はあなたの Seed から生成することができ、確実にあなたのウォレットアドレスは復元することができます。Seed からウォレットを復元するためには、"wallet with two factor authentication"を選択し、Electrum にこの特別でさまざまな Seed をウォレットの復元のために使用することを伝えます。



注意：復元オプションはあなたがもう TrustedCoin を利用したくない場合、もしくは TrustedCoin サービスに問題が生じた場合のみ、使用してください。一度この方法でウォレットを復元した場合、三つの構成要素のうち二つはあなたのマシン上にあり、2 段階認証タイプのこのウォレットの特別な保護は無効になります。

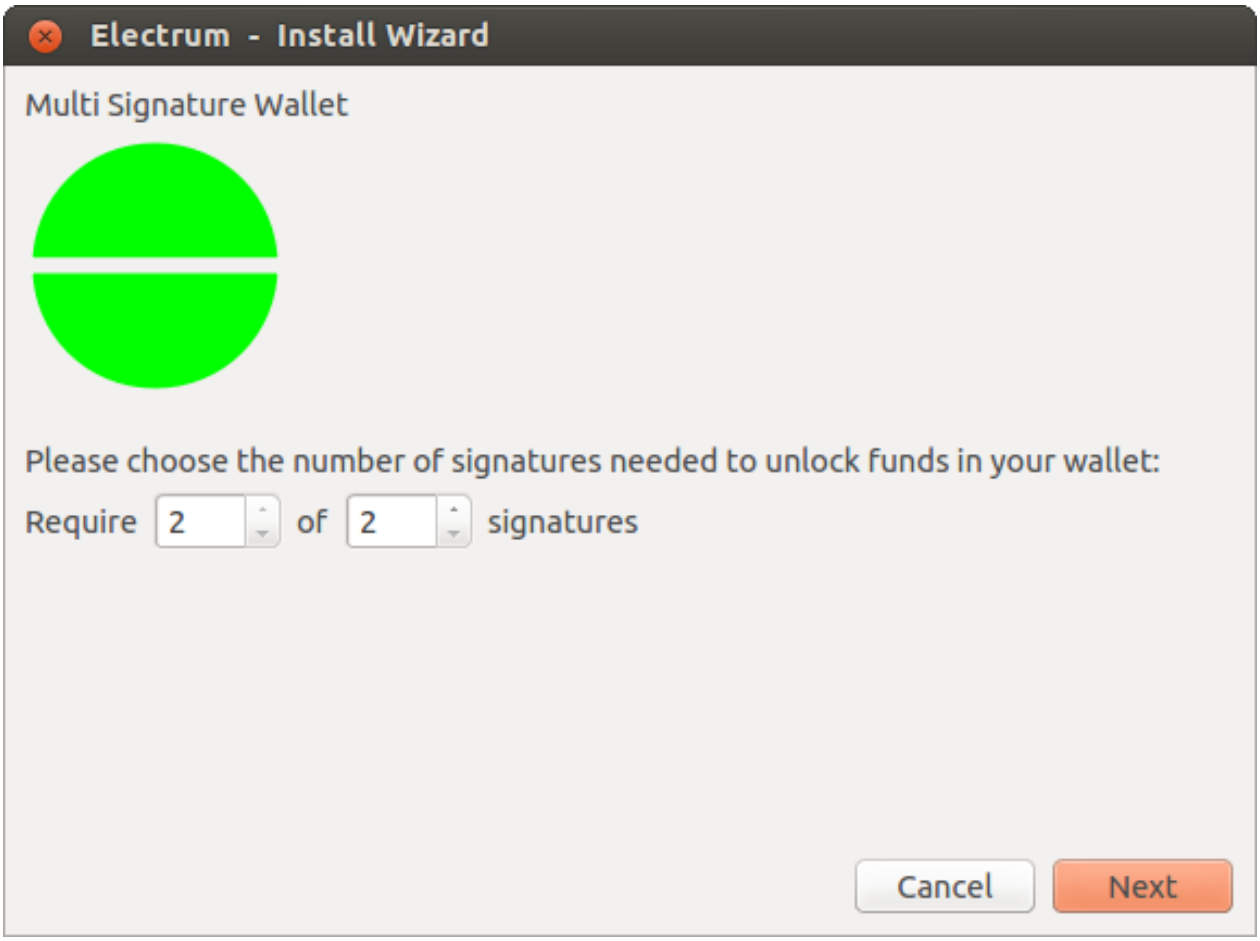
1.5 マルチシグウォレット

このチュートリアルでは 2of2 マルチシグウォレットを作成する方法をお見せします。2of2 マルチシグは 2 つの別々のウォレット（通常は別々のマシン上にありもしかすると別々の人物に管理されているかもしれない）で構成されており、資金にアクセスするためにはこれらを共に使用する必要があります。両方のウォレットに同じアドレスのセットがあります。

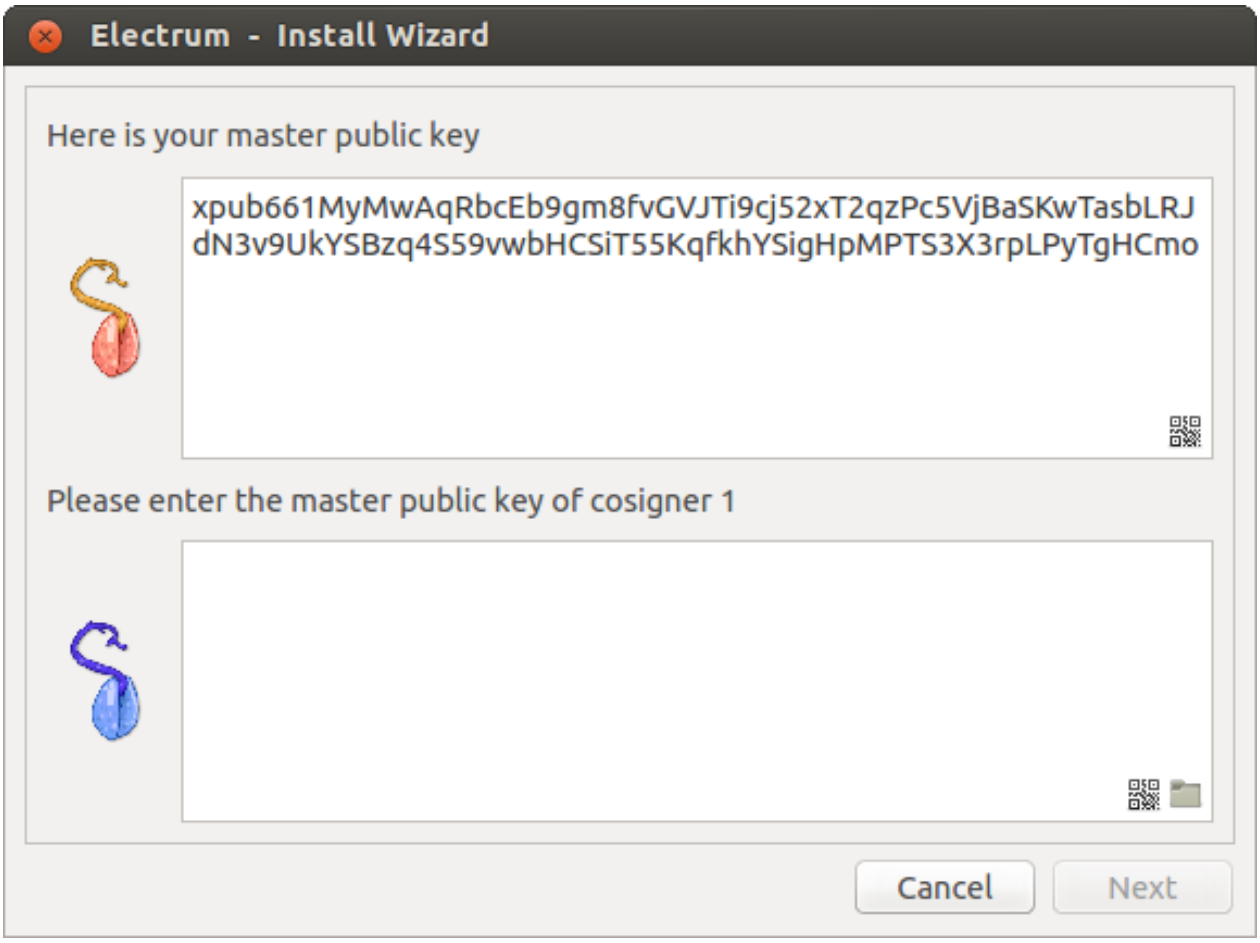
- よくある利用法として資金を共同で管理したい場合が挙げられます：もしかしたらあなたとあなたの友人が一緒に会社を経営していて、一定の資金は両方が同意しなければ使用することができないかもしれません。
- もう 1 つはセキュリティです。ウォレットの 1 つはメインマシンに、もう 1 つはオフラインマシンに置くことができます。そうすれば攻撃者やマルウェアがあなたのコインを盗むのは非常に難しくなります。

1.5.1 2-of-2 ウォレットのペアを作成する

各共同署名者 (cosigner) はこれを行う必要があります。メニューで「ファイル (File)」 - >「新規 (New)」を選択し、「Multi-signature wallet」を選択します。次の画面で、2 of 2 を選択します。



シードを生成したら（安全に保管してください）、もう一つのウォレットのマスター公開鍵を提供する必要があります。



もう一つのウォレットのマスター公開鍵を下のボックスに入れます。もちろんもう一つのウォレットを作成するときには、このウォレットのマスター公開鍵を入れます。

2 つのウォレットに対してこれを並行して実行する必要があります。このステップの間にキャンセルを押せば、ファイルを後から再度開くことができるということを留意しておいてください。

1.5.2 受信

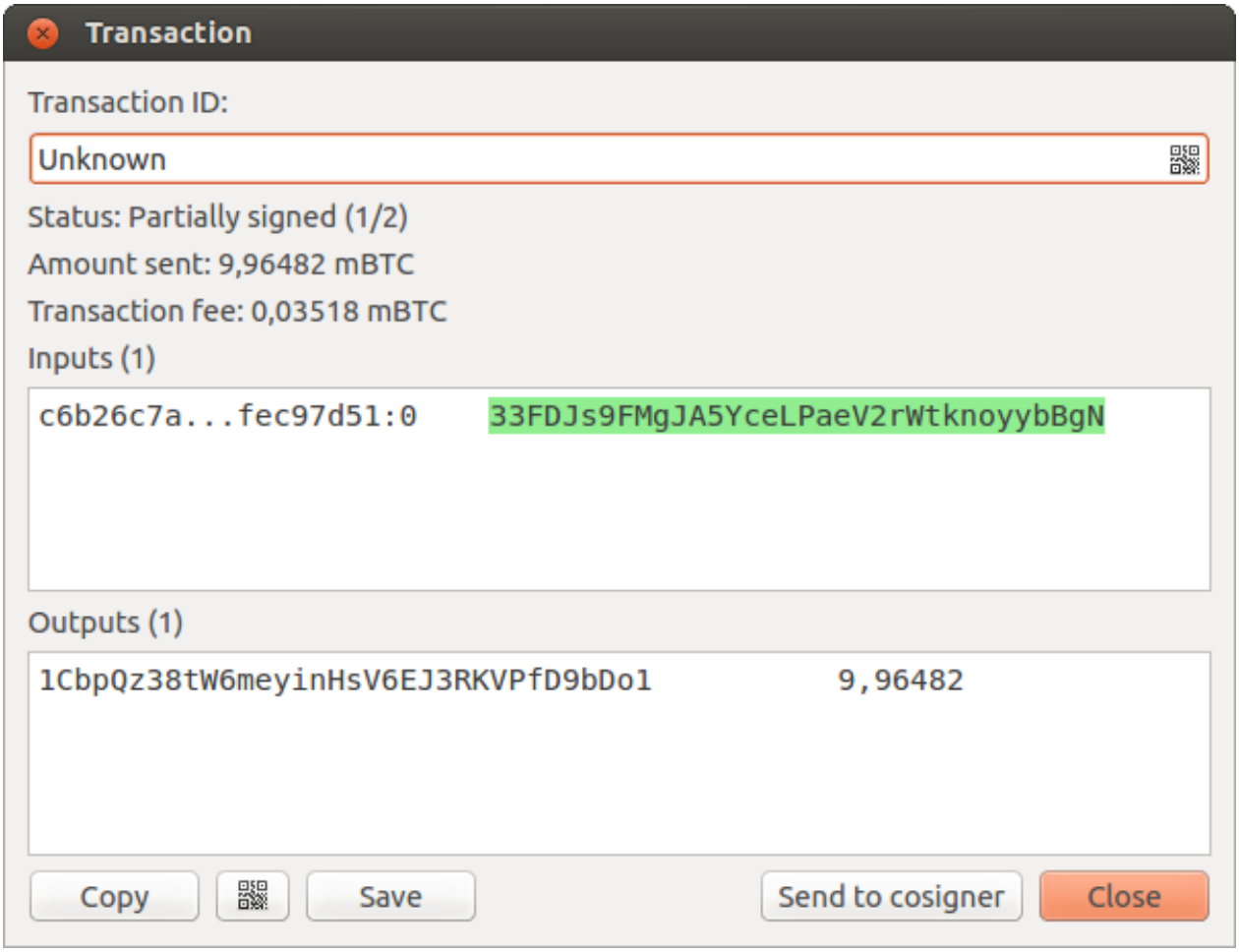
両方のウォレットが同じアドレスセットを生成していることを確認してください。あなたは P2SH アドレスに送信できるウォレットを使用してこれらのアドレス（「P」で始まることに注意）に送金できるようになりました。

1.5.3 送金

2of2 ウォレットからコインを使うには、2 人の共同署名者が協力してトランザクションに署名する必要があります。

これを達成するには、ウォレットの 1 つを使用してトランザクションを作成します（「送信 (Send)」タブのフォームを埋めてください）

署名後、トランザクションの詳細がウィンドウに表示されます。



トランザクションを 2 番目のウォレットに送る必要があります。

これには複数の選択肢があります：

- USB メモリでファイルを転送することができます
- QR コードを使用することができます
- CosignerPool プラグインを使用してリモートサーバーを使用できます

ファイルを転送する

「保存 (save)」ボタンを押して部分的に署名されたトランザクションをファイルに保存し、2 番目のウォレットが実行されているマシンに（usb メモリ等を通して）転送したら、ファイルを読み込ませましょう。（ツール (Tools)->取引情報を読み込む (Load transasction)->ファイルから (form file)）

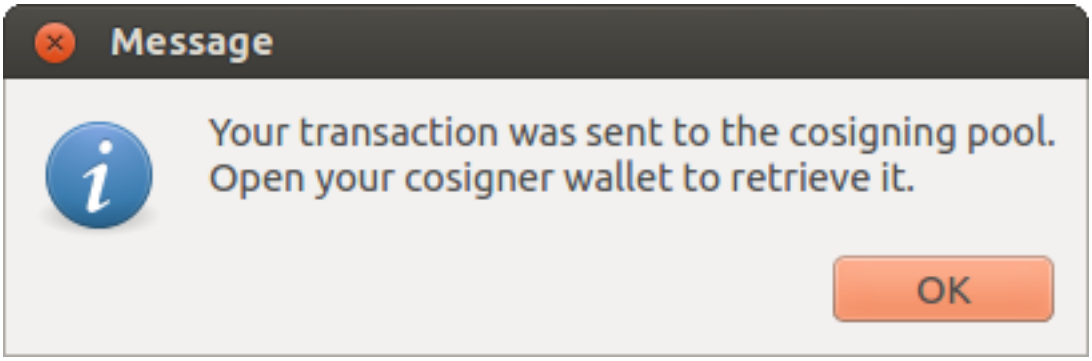
QR コードを使う

QR コードアイコンを表示するボタンもあります。これをクリックすると、2 番目のウォレットにスキャンできるトランザクションが入った QR コードが表示されます（「ツール (Tools)」->「取引情報を読み込む (Load transaction)」->「QR コードから (From QR code)」）

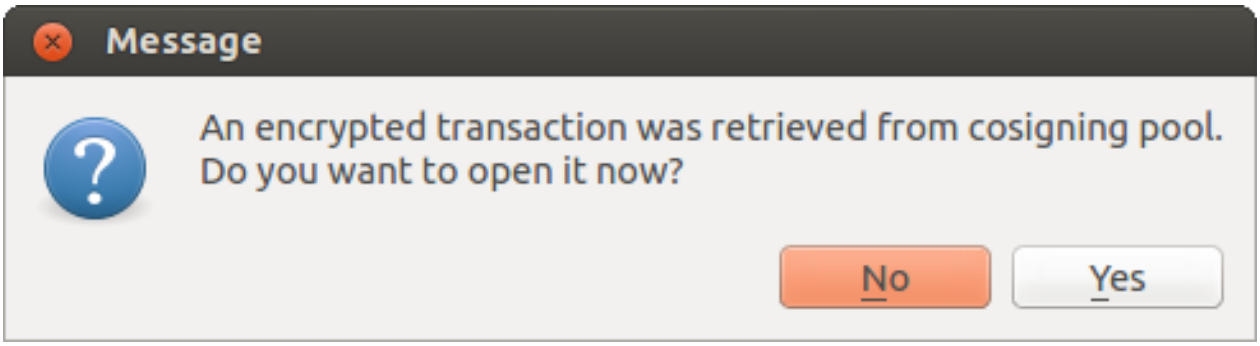
Cosiner Pool プラグインを使う

この機能のためには、両方のウォレットでプラグイン"Cosigner Pool"を有効にする必要があります（ツール (Tools)->プラグイン (Plugin)）。

プラグインが有効になると「send to cosigner」というラベルの付いたボタンが表示されます。クリックすると部分的に署名されたトランザクションが中央サーバに送信されます。トランザクションはあなたの共同署名者のマスター公開鍵で暗号化されていることに注意してください。



共同署名者のウォレットが起動すると、部分的に署名されたトランザクションが使用可能であるという通知が表示されます。



トランザクションは、共同署名者のマスター公開鍵で暗号化されているので復号するためにパスワードが必要です。

上記の順序を全て経ると、「署名」ボタンを押すことで 2 つめの署名をトランザクションに追加できるようになりました。その後、トランザクションはネットワークにブロードキャストされます。

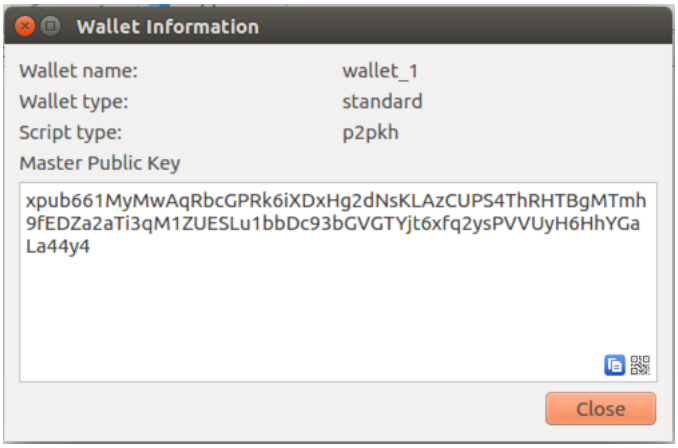
1.6 コールドストレージ

このドキュメントでは、Monacoin を保持するオフラインウォレットを作成、そしてその履歴を表示するための閲覧専用のオンラインウォレットの作成方法と、オンラインウォレットでブロードキャストする前に、オフラインウォレットでの署名が必要なトランザクションの作成方法を紹介します。

1.6.1 オフラインウォレットを作成する

通常の手順 (ファイル (File)->新規 (New)) などのようにして、オフラインのマシンでウォレットを作成します。

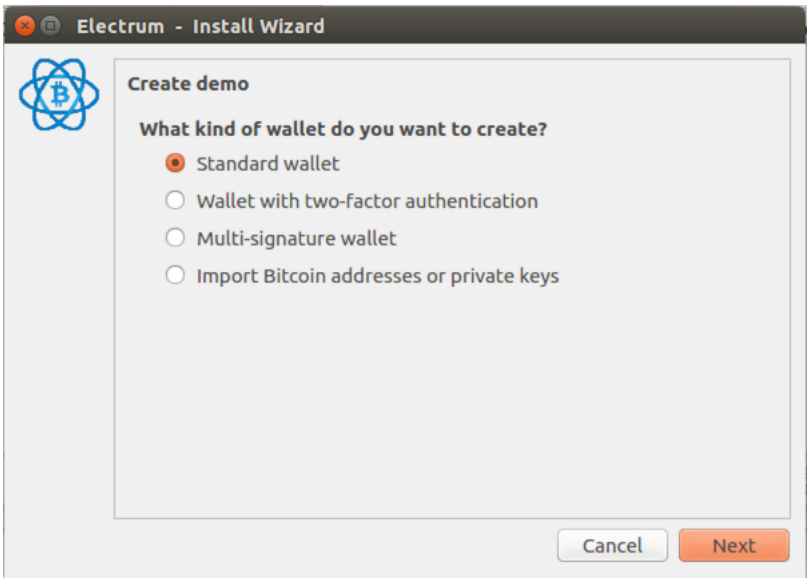
ウォレットを作成したら、ウォレット (Wallet)->情報 (Information) に移動します。



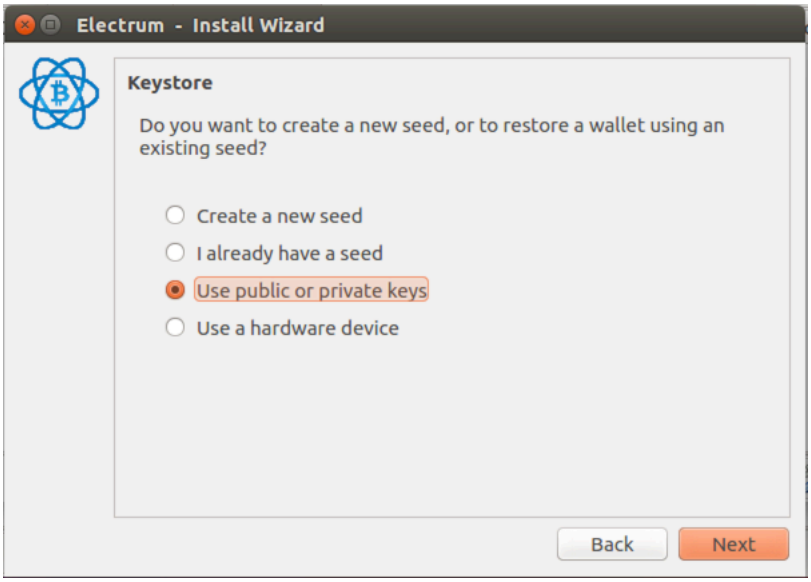
ウォレットのマスター公開鍵がポップアップウィンドウに文字列で表示されます。そのキーをあなたのオンラインマシンに何かしらの方法で移動してください。

1.6.2 ウォレットの閲覧専用バージョンを作成する

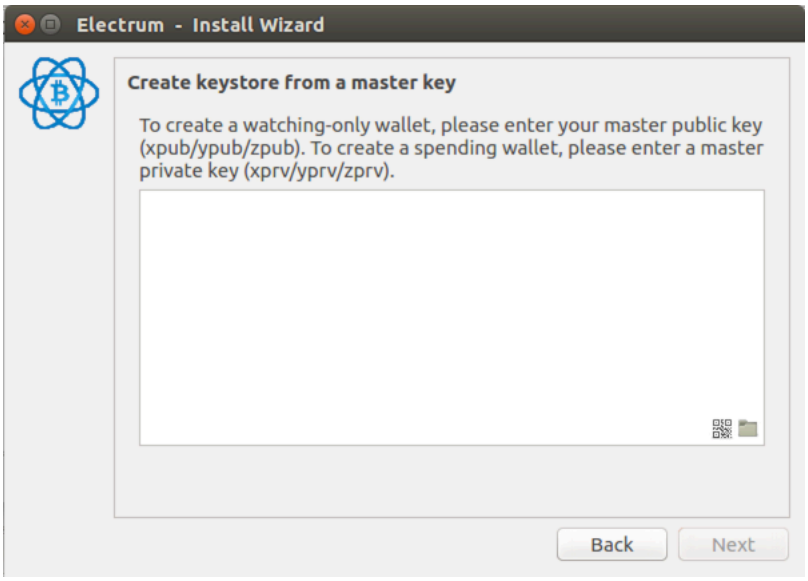
オンラインマシン上で Electrum を開き「ファイル (File)」->「新規/復元 (New/Restore)」を選択します。ウォレットの名前を入力し、「standard wallet」を選択してください。



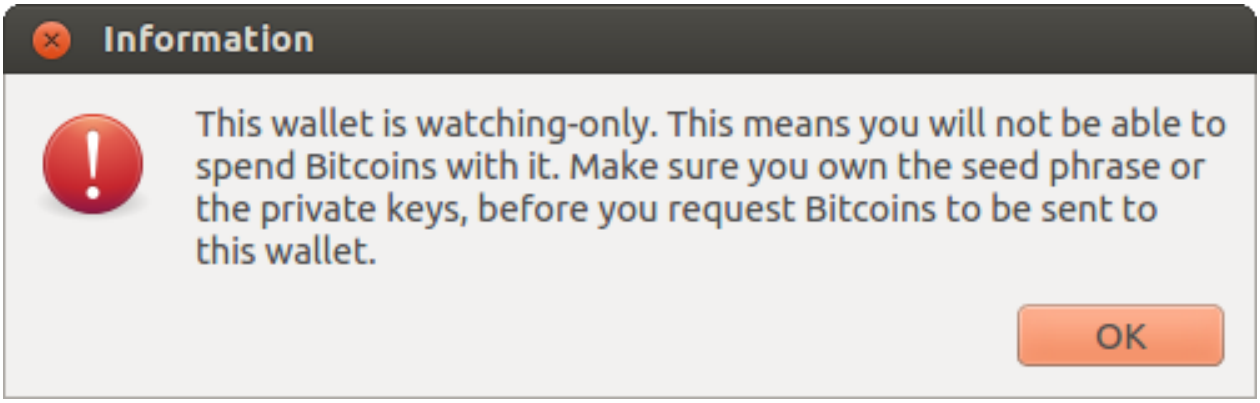
「Use public or private keys」を選択。



マスター公開鍵をボックスに貼り付けます。



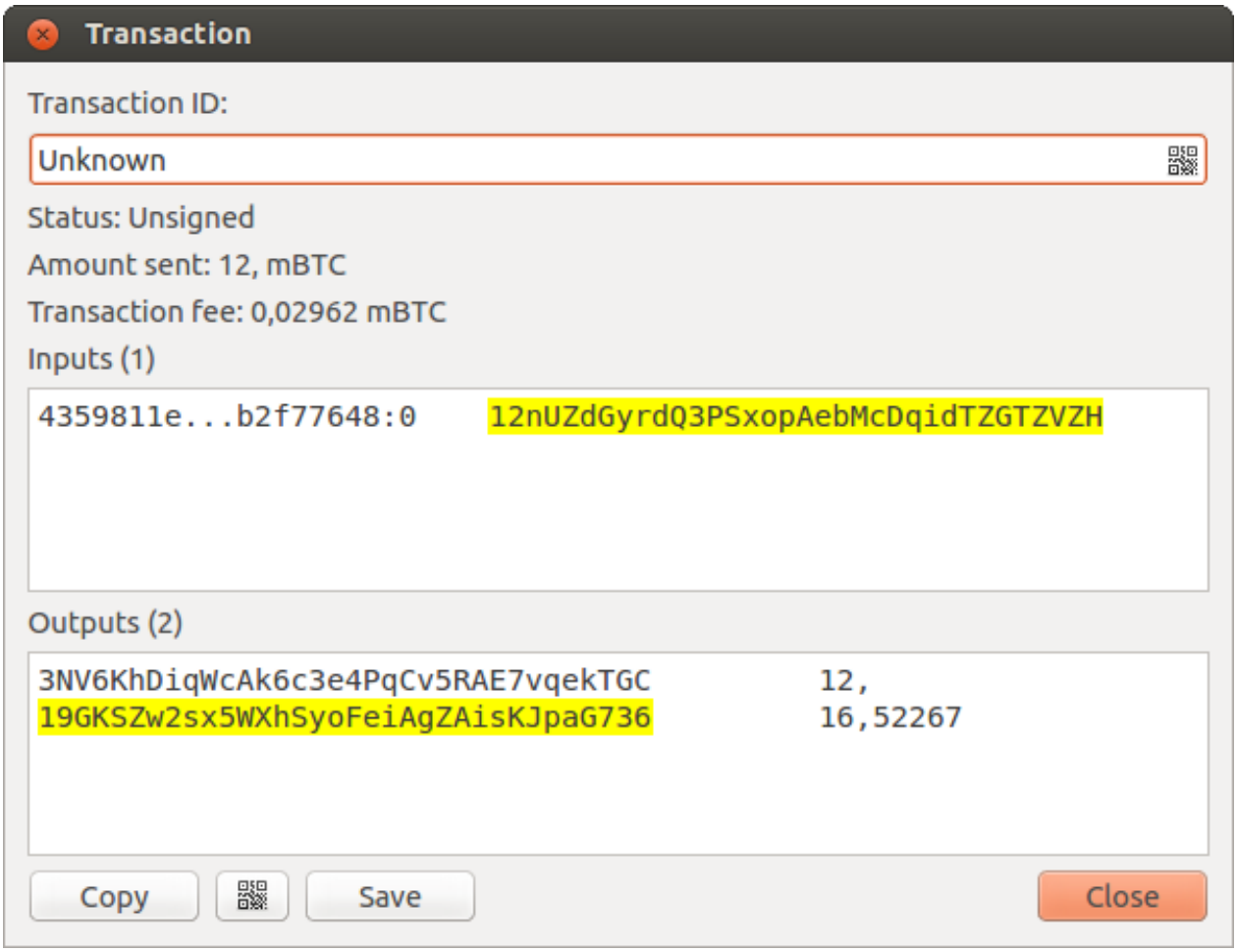
「次へ (Next)」をクリックしてウォレットの作成を完了します。作業が終わると、現在閲覧専用ウォレットを開いていることを通知するポップアップが表示されます。



その後、コールドウォレットの取引履歴が表示されます。

1.6.3 未署名のトランザクションを作成する

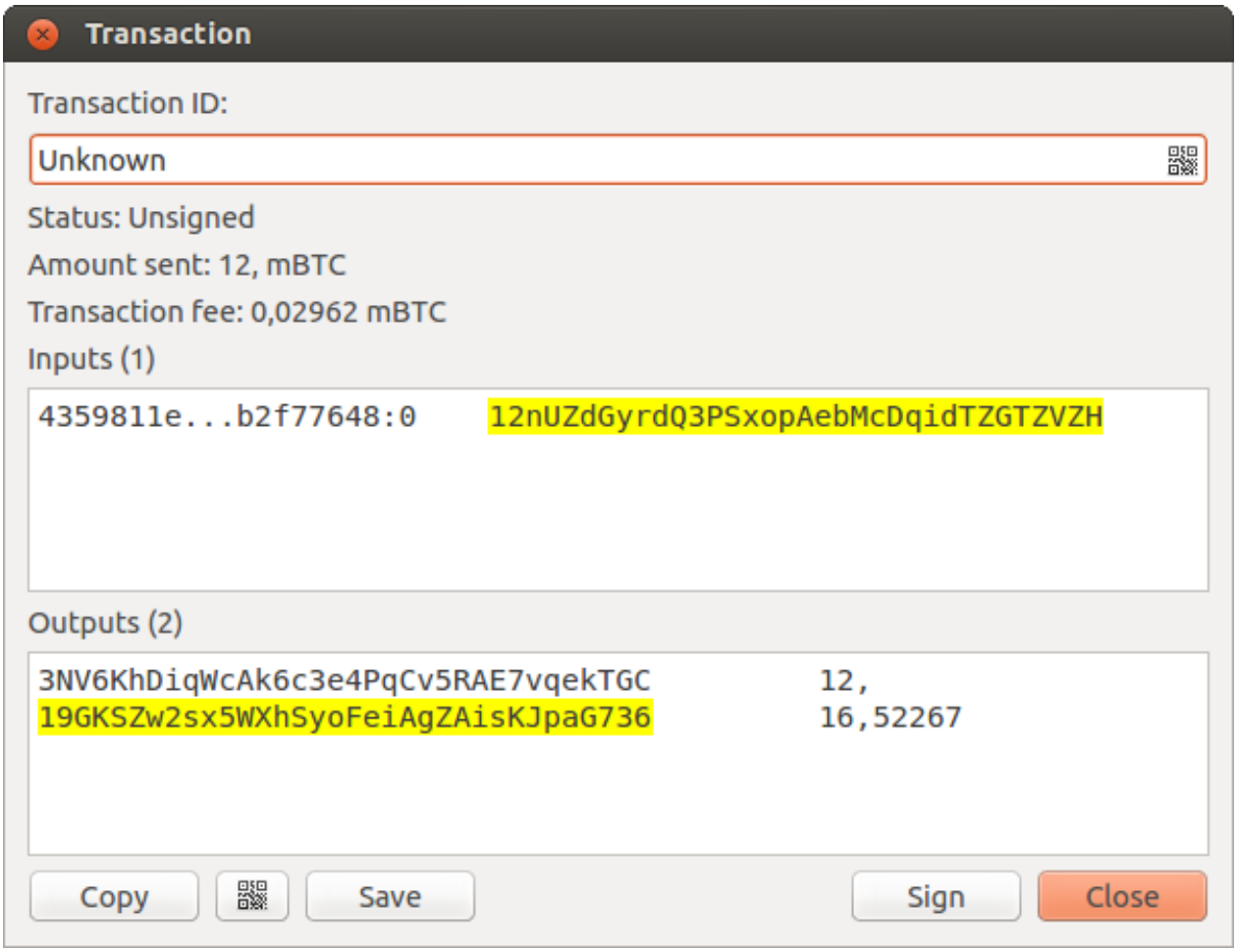
オンラインの閲覧専用ウォレットで「送信 (Send)」タブを開き、トランザクションデータを入力したら「プレビュー (Preview)」を押してください。ポップウィンドウが開きます：



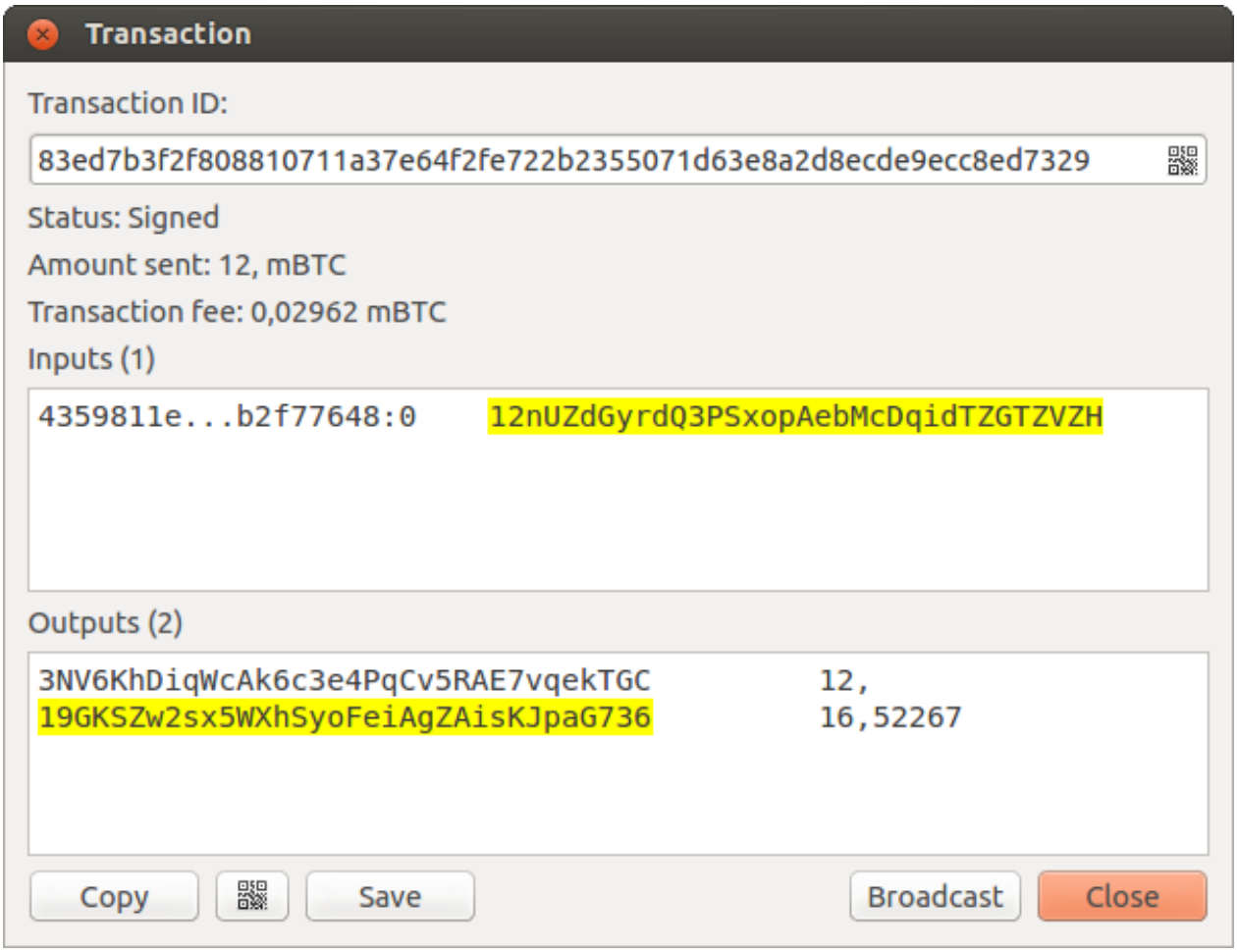
「保存 (save)」を押してコンピュータにトランザクションファイルを保存してください。ウィンドウを閉じてトランザクションファイルをあなたのオフラインマシンに転送してください。(USB メモリなどを使って)

1.6.4 トランザクションに署名する

オフラインウォレットで、メニューから「ツール (Tools)」->「取引情報の読み込み (Load transaction)」->「ファイルから (From file)」を選択し、先ほどのステップで作成したトランザクションファイルを選択します。



「署名 (sign)」を押してください。トランザクションが署名されると、トランザクション ID が所定のフィールドに表示されます。



「保存 (save)」を押して、コンピュータにファイルを保存したら、オンラインマシンにそのファイルを転送します。

1.6.5 トランザクションをブロードキャストする

オンラインマシンでメニューから「ツール (Tools)」->「取引情報の読み込み (Load transaction)」->「ファイルから (From file)」を選択します。署名済みトランザクションのファイルを選択します。開いたウィンドウで「発信 (broadcast)」を押します。トランザクションは Monacoin ネットワークを通してブロードキャストされます。

1.7 Hardware wallets on Linux

The following aims to be a concise guide of what you need to get your hardware wallet working with Electrum.

If you use the AppImage, that already has all the dependencies and Python libraries bundled with it, so skip the first two steps.

1.7.1 1. Dependencies

Currently all hardware wallets depend on `hidapi`, to be able to build that, you need:

ubuntu:

```
sudo apt-get install libusb-1.0-0-dev libudev-dev
```

fedora:

```
sudo dnf install libusb-devel systemd-devel
```

(Package names may be different for other distributions.)

1.7.2 2. Python libraries

Then depending on the device you have, you need a python package (typically a library by the manufacturer):

Trezor

```
python3 -m pip install trezor[hidapi]
```

For more details, refer to python-trezor.

Ledger

```
python3 -m pip install btchip-python
```

For more details, refer to btchip-python.

KeepKey

```
python3 -m pip install keepkey
```

For more details, refer to python-keepkey.

Digital Bitbox

The Digital Bitbox only needs `hidapi`.

```
python3 -m pip install hidapi
```

Archos Safe-T

```
python3 -m pip install safet
```

For more details, refer to python-safet.

Coldcard

```
python3 -m pip install ckcc-protocol
```

For more details, refer to ckcc-protocol.

1.7.3 3. udev rules

You will need to configure udev rules:

Trezor

See TREZOR User Manual: Configuration of udev rules

backup link

Ledger

See Ledger Support Center: What to do if my Ledger Nano S is not recognized on Windows and/ or Linux?

backup link

KeepKey

See KeepKey Support Desk: KeepKey wallet is not being recognized by Linux

backup link

Digital Bitbox

See Bitbox | Linux

Archos Safe-T

See this file in their GitHub repository.

Coldcard

See this file in their GitHub repository.

(It should go into `/etc/udev/rules.d/51-coinkite.rules` or `/usr/lib/udev/rules.d/51-coinkite.rules`)

1.7.4 4. Apply configuration

To apply the changes, reload udev rules (or reboot):

```
sudo udevadm control --reload-rules && sudo udevadm trigger
```

1.7.5 5. Done

That ' s it! Electrum should now detect your device.

1.8 Using the most current Electrum on Tails

Tails currently ships with an outdated version of Electrum that can no longer be used on the public Electrum server network as of March 2019. Unfortunately installed software on Tails cannot be permanently upgraded since it is a fixed (read-only) image - you have to wait for Tails to upgrade.

However, you can use a recent version of Electrum with Tails by using the AppImage binary we distribute (a self-contained executable that works on x86_64 Linux including Tails).

These steps have been tested on Tails 3.12.1 and 3.13.

1.8.1 Steps to use AppImage

1. Write down your wallet seed words and store them securely off the computer.
2. Enable and configure persistent storage. In Tails enter the Applications/Tails menu and select "Configure persistent volume". Ensure "Personal data" and "Bitcoin client" sliders are enabled. Reboot if necessary and make sure the persistent volume is unlocked.
3. Ensure your Tails is connected to a WiFi network and the onion icon at the top confirms Tor network is ready.

4. Using Tor browser download the Linux Appimage file under "Sources and Binaries" near the top of the download page on electrum.org and save it to the default "Tor browser" folder. Tails/Tor are not as fast as your regular computer OS/WiFi and the download may take much longer than normally expected, especially if you have a slow computer or USB drive. Tor download speed depends entirely on the Tor network connections.

5. Open Home/Tor browser folder and drag appimage to the Persistent folder (lower left side of the window). Tails is very sensitive to user writeable file locations and Electrum may not work in another location.

6. Open Home/Persistent folder (where the appimage will now live), right click on the appimage, select permissions tab and click "Allow executing file as program" then close the dialog. More detailed instructions with screenshots are available here.

7. Optional: Check the PGP signature of the AppImage by following these instructions before using the AppImage.

You can now simply click on the appimage icon in your persistent folder to run the newest Electrum. Your wallet can be recreated by entering the seed words when prompted. This image and any data (wallets) it creates will remain on your Tails USB drive as long as you've saved it to persistent storage.

Caution: Do not use the old Electrum availble in the Tails menus. Your new AppImage *should* use the previous persistent Electrum data without difficulty. If there is any question about wallets being corrupted erase the files in ~/.electrum/wallets and reinitialize them from seed words.

第 2 章

Advanced users

2.1 コマンドライン

Electrum には強力なコマンドラインがあります。このページではいくつかの基本原則を示します。

2.1.1 インラインヘルプの使用

Electrum のコマンドリストを表示するには、次のように入力します。

```
electrum help
```

コマンドのドキュメントを表示するには、次のように入力します。

```
electrum help <command>
```

2.1.2 マジックワード

コマンドに渡す引数には、次のマジックワードを使うことがあります：!?-

- エクスクラメーションマーク"!"は利用可能な最大の金額を意味するショートカットです。

  例：

  ```
  electrum payto 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE !
  ```

  トランザクション手数料は計算され、総計から差し引かれることに気を付けてください。

- クエスチョンマーク"?"はそのパラメータを入力してもらいたいことを表すためのものです。

  例：

```
electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE ?
```

- コロン":"は入力したパラメータを隠したいとき（ターミナルに表示したくないとき）に使います。

```
electrum importprivkey :
```

この例では、秘密鍵とウォレット・パスワードの 2 つのプロンプトが表示されます。

- ダッシュ"-"で置き換えられたパラメータは、標準入力（パイプ内）から読み込まれます。

```
cat LICENCE | electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE -
```

2.1.3 エイリアス

ほとんどのコマンドにおいて、Monacoin アドレスの部分に DNS エイリアスを使うことできます。

```
electrum payto ecdsa.net !
```

2.1.4 jq を使って出力を成形する。

コマンド出力はシンプルな文字列か json 形式のデータのどちらかになっています。jq プログラムというとても便利なユーティリティががあります。次のようにしてインストールできます。

```
sudo apt-get install jq
```

以下の例では jp プログラムを用いています。

2.1.5 例

メッセージの署名と検証

変数を使用して署名を格納し、検証することができます。

```
sig=$(cat LICENCE| electrum signmessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE -)
```

そして：

```
cat LICENCE | electrum verifymessage 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE $sig -
```

未使用値を表示する。

'listunspent' コマンドは様々なフィールドを持つ辞書オブジェクトのリストを返します。各レコードの 'value' フィールドを抽出したいとすると、これは jp コマンドを用いることで可能です。

```
electrum listunspent | jq 'map(.value)'
```

履歴から受信トランザクションのみを選択する。

受信トランザクションは正の'value' フィールドを持ちます。

```
electrum history | jq '.[] | select(.value>0)'
```

日付によってトランザクションをフィルタリングする

次のコマンドは、指定された日付の後にタイムスタンプされたトランザクションを選択します。

```
after=$(date -d '07/01/2015' +"%s")

electrum history | jq --arg after $after '.[] | select(.timestamp>($after|tonumber))'
```

同様に、一定期間のトランザクションをエクスポートすることができます。

```
before=$(date -d '08/01/2015' +"%s")

after=$(date -d '07/01/2015' +"%s")

electrum history | jq --arg before $before --arg after $after '.[] | select(.timestamp&
↪gt;($after|tonumber) and .timestamp&lt;($before|tonumber))'
```

メッセージの暗号化と復号化

最初にウォレットアドレスの公開鍵が必要です。

```
pk=$(electrum getpubkeys 1JuiT4dM65d8vBt8qUYamnDmAMJ4MjjxRE| jq -r '.[0]')
```

暗号化：

```
cat | electrum encrypt $pk -
```

復号化：

```
electrum decrypt $pk ?
```

注：このコマンドは暗号化済みメッセージを要求し、その際にウォレットのパスワードを要求します。

秘密鍵のエクスポートとコインのスイープ

次のコマンドでウォレットの Monacoin を持つすべてのアドレスをエクスポートします。

```
electrum listaddresses --funded | electrum getprivatekeys -
```

これによって秘密鍵のリストが返されます。多くの場合ではシンプルなリストが欲しいでしょう。これは jq フィルターを追加することで可能です。

```
electrum listaddresses --funded | electrum getprivatekeys - | jq 'map(.[0])'
```

最後に、この秘密鍵のリストを使用して sweep コマンドに入力してみましょう。

```
electrum listaddresses --funded | electrum getprivatekeys - | jq 'map(.[0])' |␣
↪electrum sweep - [destination address]
```

2.2 コマンドラインからコールドストレージを使用する

このページではコマンドラインを使って、オフラインの Electrum ウォレットでトランザクションに署名する方法を説明します。

2.2.1 未署名のトランザクションを作成する

オンライン（閲覧専用）ウォレットを使って、未署名のトランザクションを作成します。

```
electrum payto 1Cpf9zb5Rm5Z5qmmGezn6ERxFWvwuZ6UCx 0.1 --unsigned > unsigned.txn
```

未署名のトランザクションは、 'unsigned.txn' という名前のファイルに格納されます。閲覧専用ウォレットを使用する場合は、–unsigned オプションは不要です。

以下のコマンドで中身を見ることができます。

```
cat unsigned.txn | electrum deserialize -
```

2.2.2 トランザクションに署名する

Electrum のシリアライズ形式には、オフラインウォレットでトランザクションに署名する際に使用される、必要なマスター公開鍵と鍵導出 (Key derivation) が含まれています。

したがって、シリアライズされたトランザクションをオフラインウォレットに渡すだけで済みます。

```
cat unsigned.txn | electrum signtransaction - > signed.txn
```

このコマンドではあなたのパスワードが求められ、署名されたトランザクションは'signed.txn' に保存されます。

2.2.3 トランザクションをブロードキャストする

'broadcast' を使用して Monacoin ネットワークにトランザクションを送信します。

```
cat signed.txn | electrum broadcast -
```

成功すると、コマンドはトランザクション ID を返します。

2.3 Electrum を使った Web サイト上での Monacoin の受け取り方

このチュートリアルでは SSL 化された支払いリクエストを使って Web サイト上で BIP-70 に基づいた Monacoin を受け取る方法を説明しています。Electrum2.6. で更新されています。このドキュメントは 3.1.2 で更新されています。

2.3.1 要件

- static HTML を提供する Web サーバ
- SSL 証明書（CA(認証局) による署名 例えば Letsencrypt）
- バージョン 3.1 以上の Electrum
- Electrum-Merchant

2.3.2 商業用のウォレットを作成、使用する

暗号通貨を安全に保存しておきたい場合は、保護されたマシン上にウォレットを作成してください。もしあなたの商業用サーバに誰かが侵入した場合、彼又は彼女は読み取り専用のウォレットにアクセスすることができるだけでコインを使用することはできません。しかし内部に隠れた侵入者は依然としてあなたのアドレス、取引、残高を監視することができます。商業目的には（あなたのメインウォレットではなく）分けたウォレットを使用することが推奨されています。

```
electrum create
```

保護されたマシン上でマスター公開鍵 (xpub) をエクスポートする：

```
electrum getmpk -w .electrum/wallets/your-wallet
```

これで商業用 Electrum デーモンを設定することができるようになりました。

サーバマシン上で先ほどエクスポートしたマスター公開鍵 (xpub) を復旧します。

```
electrum restore xpub.....................................................
```

あなたの読み取り専用のウォレットを（再）復旧したら Electrum をデーモンとして起動します：

```
electrum daemon start
electrum daemon load_wallet
```

2.3.3 設定に SSL 証明書を追加する

あなたのドメイン用の秘密鍵と公開証明書を持っている必要があります。これはウォレット・キーではなく、一致する TLS/SSL 証明書の秘密キーであることに注意してください。

秘密鍵だけを含んだファイルを作成してください。

```
-----BEGIN PRIVATE KEY-----
your private key
-----END PRIVATE KEY-----
```

setconfig コマンドで秘密鍵のファイルへのパスを指定してください。

```
electrum setconfig ssl_privkey /path/to/ssl.key
```

別のファイルを作成し、あなたの証明書とルートの認証局まで依存する証明書のリストを含めてください。あなたの証明書はリストの最初、ルートの認証局はリスト最後にある必要があります。

```
-----BEGIN CERTIFICATE-----
your cert
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
intermediate cert
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
root cert
-----END CERTIFICATE-----
```

setconfig コマンドで ssl_chain までのパスを指定してください。

```
electrum setconfig ssl_chain /path/to/ssl.chain
```

2.3.4 リクエストディレクトリを設定する

このディレクトリはあなたの Web サーバに設置されなければなりません。（Apache や Nginx 等）

```
electrum setconfig requests_dir /srv/www/payment/
```

デフォルトでは Electrum は'file:///' で始まるローカルの URL を表示します。公開された URL を表示するためには、url_rewrite を設定する必要があります。例えば、

```
electrum setconfig url_rewrite "[ 'file:///srv/www/', 'https://example.com/' ]"
```

Web サーバーは、支払要求を処理できるように準備しておく必要があります。configuration for Nginx:

```
location /payment/ {
    default_type "application/bitcoin-paymentrequest";
    alias /srv/www/payment/;
}
```

Or for Apache 2:

```
<Directory /srv/www/payment/requests/>
    ForceType application/bitcoin-paymentrequest
</Directory>
```

その他の Web サーバーについては、ディレクトリに基づいてデフォルトの MIME タイプを変更する方法について、マニュアルを参照してください。

2.3.5 Electrum-Merchant をインストールする

Electrum-Merchant をインストールして走らせるプログラム。 デフォルトでは、単純なインタフェースがインストールされます。他のインタフェースは準備中であり、将来利用可能になる予定です。

```
pip3 install electrum-merchant
python3 -m electrum-merchant
```

Electrum-Merchant を正常に実行するには、前述の手順に従う必要があることに注意してください。

2.3.6 署名された支払いリクエストを作成する

```
electrum addrequest 3.14 -m "this is a test"
{
   "URI": "bitcoin:1MP49h5fbfLXiFpomsXeqJHGHUfNf3mCo4?amount=3.14&r=https://example.
↪com/payment/7c2888541a",
```

(次のページに続く)

(前のページからの続き)


```
    "address": "1MP49h5fbfLXiFpomsXeqJHGHUfNf3mCo4",
    "amount": 314000000,
    "amount (BTC)": "3.14",
    "exp": 3600,
    "id": "7c2888541a",
    "index_url": "https://example.com/payment/index.html?id=7c2888541a",
    "memo": "this is a test",
    "request_url": "https://example.com/payment/7c2888541a",
    "status": "Pending",
    "time": 1450175741
}
```


このコマンドは二つの URL とともに json オブジェクトを返します：

- request_url は署名された BIP70 リクエストの URL
- index_url はリクエストを表示する Web ページの URL

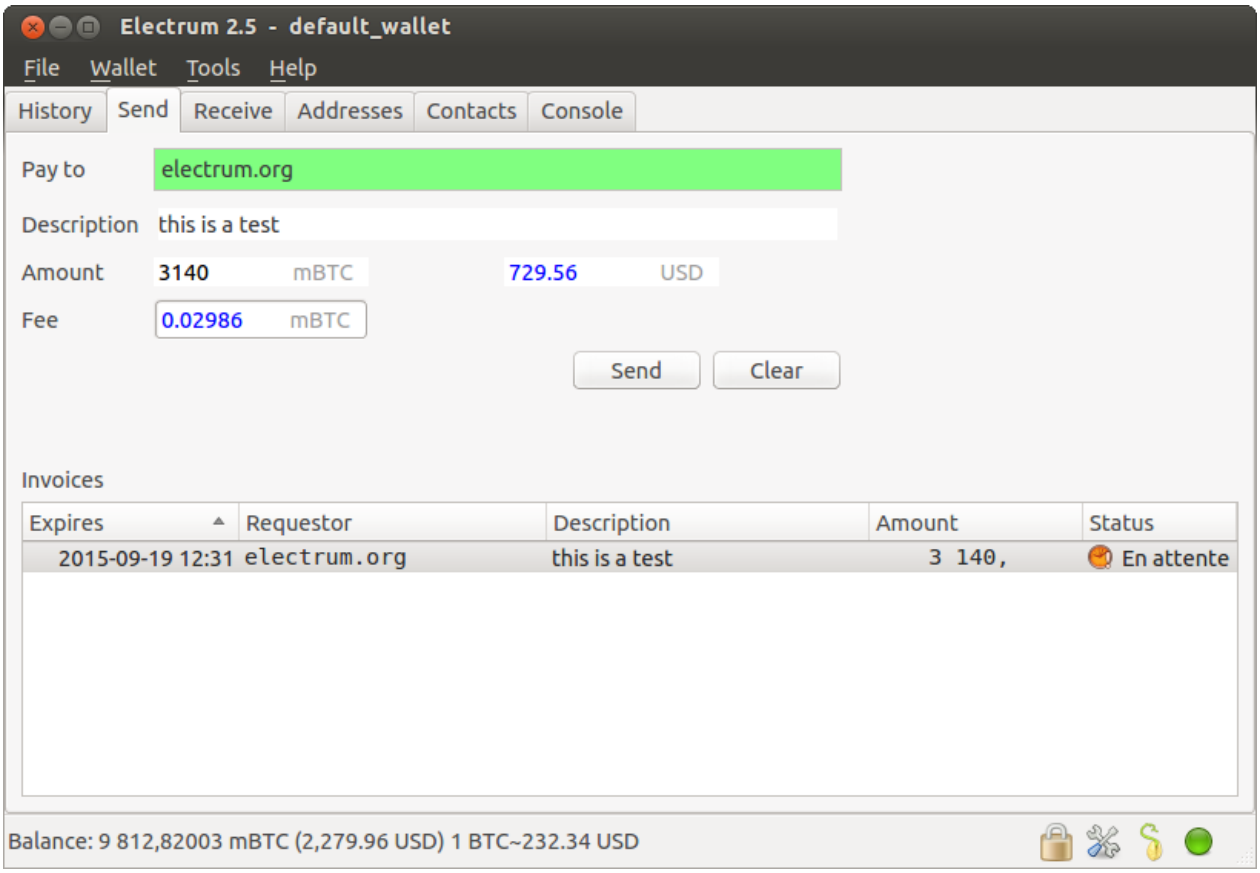
request_url と index_url は url_rewrite に定義したドメイン名を使用することに気を付けてください。

'listrequests' コマンドを使用することで現在のリクエストの一覧を閲覧することができます。

2.3.7 ブラウザで支払いリクエストのページを開く

index_url を Web ブラウザで開いてみましょう。

このページには支払いリクエストが載っています。ウォレットで Monaocin: URI を開くか、QR コードをスキャンすることが出来ます。最後の行はリクエストの期限が切れるまでに残された時間を表しています。



既にこのページは支払いを受け取るために使用できます。ただしリクエストが支払われたかどうかは検出しません。そのためには Web ソケットを設定する必要があります。

2.3.8 Web ソケットのサポートを追加する

SimpleWebSocketServer をここから入手してください：

```
git clone https://github.com/dpallot/simple-websocket-server
```

設定に ‘‘websocket_server‘‘ と Âă‘‘websocket_port‘‘を指定してください：

```
electrum setconfig websocket_server example.com

electrum setconfig websocket_port 9999
```

デーモンを再起動します：

```
electrum daemon stop

electrum daemon start
```

これでページは完全に対話的になり、支払いを受け取るとページが更新されます。

クライアントのファイアウォールで上位のポートがブロックされる可能性があることに注意してください。そのため、たとえば追加の外部 IP アドレスで標準の 443 ポートを使用してプロキシ Web ソケットの送信を逆にする方が安全です。ローカルにインストールされている Websocket サーバまたはサーバのポート Electrum サーバが、カスタマに通知するポートと異なる場合は、次の 2 つの追加設定パラメータを設定できます。

- websocket_server_announce
- websocket_port_announce

これでこのページは完全にインタラクティブになり、支払いを受け取ると自らアップデートするようになりました。higher port は一部のクライアントのファイアウォールにブロックされる可能性があるので、たとえば追加のサブドメイン上の標準ポートである 443 を使用して Web ソケットの転送をリバースプロキシで行う方が安全です。

2.3.9 JSONRPC インターフェイス

Electrum デーモンへのコマンドは JSONRPC を使用して送ることができます。PHP スクリプトで Elecrum を使用したいときに役に立つでしょう。

デーモンはデフォルトではランダムなポート番号を使うことに気を付けてください。確実なポート番号を使うには、'rpcport' 設定値を指定（してデーモンを再起動）する必要があります。：

```
electrum setconfig rpcport 7777
```

さらに Electrum 3.0.5 以降、JSON-RPC インターフェースは HTTP Basic 認証を使用して認証されます。

ユーザー名とパスワードは設定変数です。最初に起動するとき、Electrum は両方を初期化し、パスワードはランダムな文字列に設定されます。後からそれらを変更することもできます（ポートと同じ方法、完了したらデーモンを再起動してください）。それらの値を簡単に見るには、

```
electrum getconfig rpcuser
electrum getconfig rpcpassword
```

HTTP Basic 認証はリクエストの一部として、暗号化されていないユーザー名とパスワードを送信することに注意してください。我々の見解として localhost 上での使用は結構ですが、信頼できない LAN やインターネット上での使用は安全ではありません。そのためセキュアなトンネルで接続をラップするなど、そういった場合にはさらなる対策を講じる必要があります。詳細については、こちらをお読みください。

静的ポートを設定し、認証を設定したら、curl または PHP を使用してクエリを実行できます。例：

```
curl --data-binary '{"id":"curltext","method":"getbalance","params":[]}' http://
↪username:password@127.0.0.1:7777
```

名前付きパラメータを使用したクエリ：

```
curl --data-binary '{"id":"curltext","method":"listaddresses","params":{"funded":true}}
↪' http://username:password@127.0.0.1:7777
```

支払いリクエストを作成する：

```
curl --data-binary '{"id":"curltext","method":"addrequest","params":{"amount":"3.14",
↪"memo":"test"}}' http://username:password@127.0.0.1:7777
```

2.4 フォークが発生した場合の Electrum を用いたコインの分離方法

2.4.1 注意：

この文書は Electrum2.9 用に更新されています。

2.4.2 フォークとは何ですか？

ブロックチェーンのフォーク（分岐）は、逸脱するネットワークがオリジナルのチェーンから枝分かれして競合するブロックのチェーンが生成、維持されると発生します。本質的に Monacoin（暗号通貨）の別のバージョンが生まれ、独自のブロックチェーン、ルールセット、市場価値を持ちます。

Monacoin ブロックチェーンのフォークが発生した場合、二つの別々の通貨が同時に存在し、異なる市場価値を持ちます。

2.4.3 コインを分離するとはどういう意味ですか？

オリジナルブロックチェーン上のアドレスには新しいチェーン上での同じ金額が含まれています。

フォーク以前に Monacoin を所持していた場合、フォーク後にこれらのコインを使用するトランザクションは一般的には両方のチェーンで有効です。つまり、あなたは両方のコインを同時に使用するかもしれないということです。これは「リプレイ」と呼ばれています。これを防ぐには、両方のチェーンで異なるトランザクションを使用してコインを移動させる必要があります。

2.4.4 フォークの検出

Electrum(バージョン 2.9 以上) はサーバ間のコンセンサス障害（ブロックチェーンのフォーク）を検出し、ユーザーがフォークのブランチを選択できるようにします。

- Electrum は Monacoin ブロックチェーンにおけるフォークの、異なるブランチに追従しているかもしれない複数のサーバから送信されるブロックヘッダをダウンロードし検証します。線形性シーケンスの代わりにブ

ロックヘッダは木構造で編成されます。分岐点は、バイナリ検索を使用して効率的に配置されます。MCV の目的は、古典的な SPV モデルでは見えないブロックチェーンフォークを検出して処理することです。

- ブロックチェーンフォークの希望のブランチはネットワークダイヤログを使用して選択できます。ブランチは、分岐ブロックのハッシュと高さによって識別されます。RBF トランザクションを使用してコインの分離が可能です（チュートリアルが追加されます)。

この機能を使用すると、あなたがコインを消費するチェーンとネットワークを選別することができます。

2.4.5 手順

1. 準備

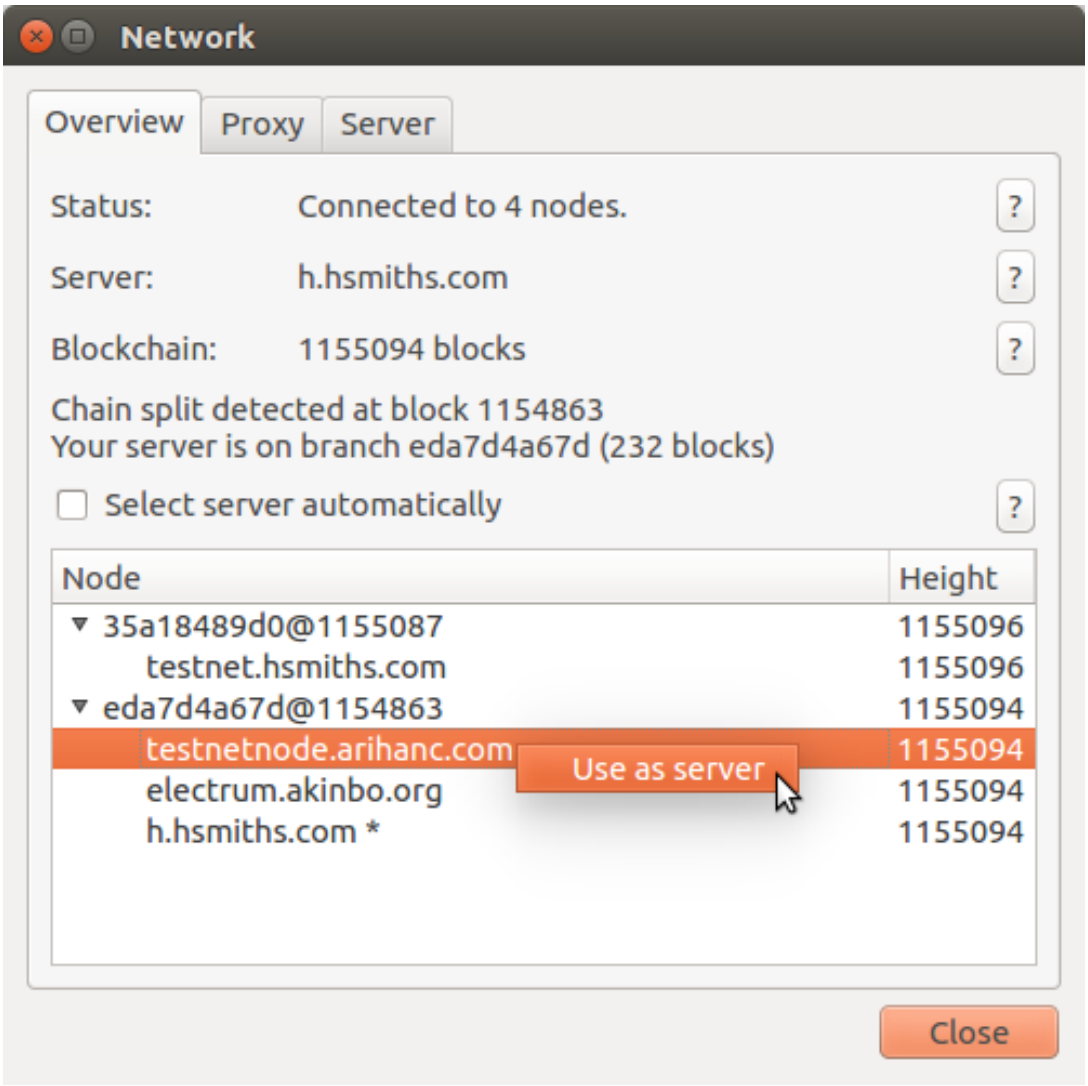
   a. メニュー âđđ 表示 âđđ 表示 コイン

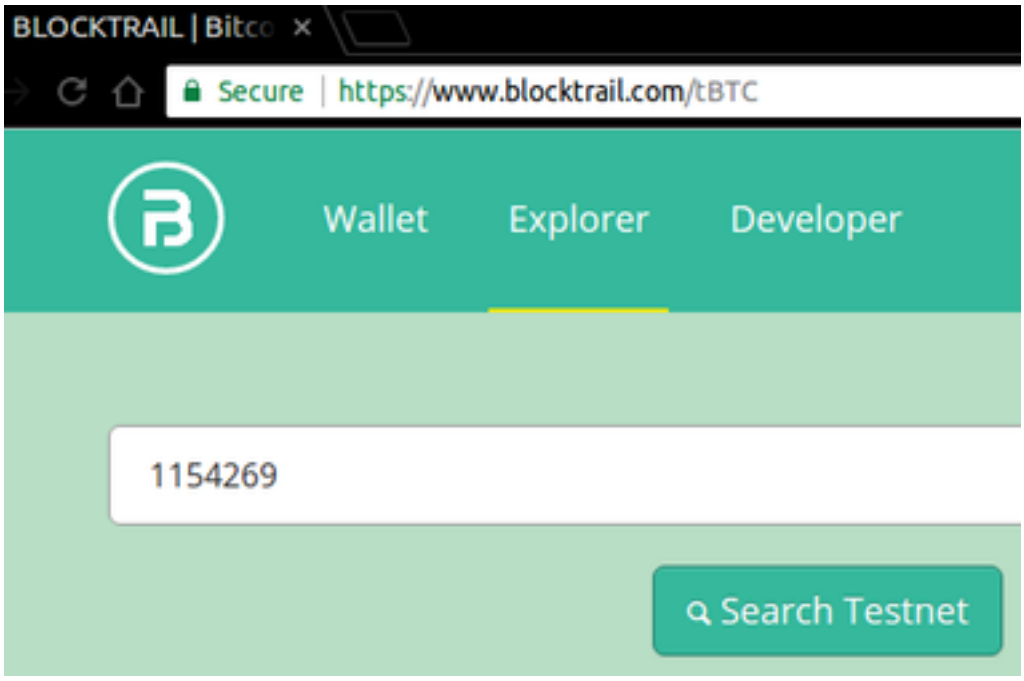
   b. メニュー âđđ ツール âđđ 設定 âđđ Propose Replace-By-Fee âđđ "Always"

2. チェーン/ネットワークを選択する

   a. メニュー âđđ ツール Âăâđđ ネットワーク

   Âă Âă ブランチは異なる高さでは異なるハッシュを持つことに注意してください。ブロックエクスプローラーを使ってハッシュと高さを確認することであなたが度のチェーン上にいるか確認することができます。


Network
Overview
Proxy
Server
Status:
Connected to 4 nodes.
Server:
h.hsmiths.com
Blockchain:
1155094 blocks
Chain split detected at block 1154863
Your server is on branch eda7d4a67d (232 blocks)
Select server automatically
Node
Height
35a18489d0@1155087
1155096
testnet.hsmiths.com
1155096
eda7d4a67d@1154863
1155094
testnetnode.arihanc.com
1155094
Use as server
electrum.akinbo.org
1155094
h.hsmiths.com *
1155094
Close



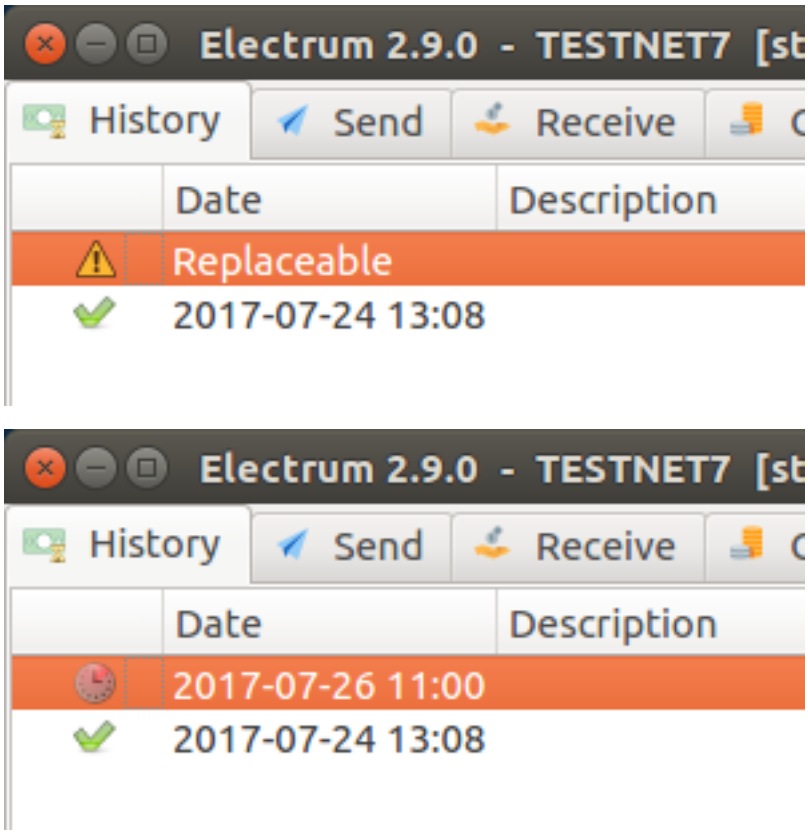
3. あなた自身にコインを送信する

Âă a. 受信アドレスを送信タブにコピー Âă b. 分離したいコインの数を入力し素（全ての場合"!"を入力）Âă c. "Replaceable"をチェック Âă d. 送信 âđđ 署名 âđđ 発信

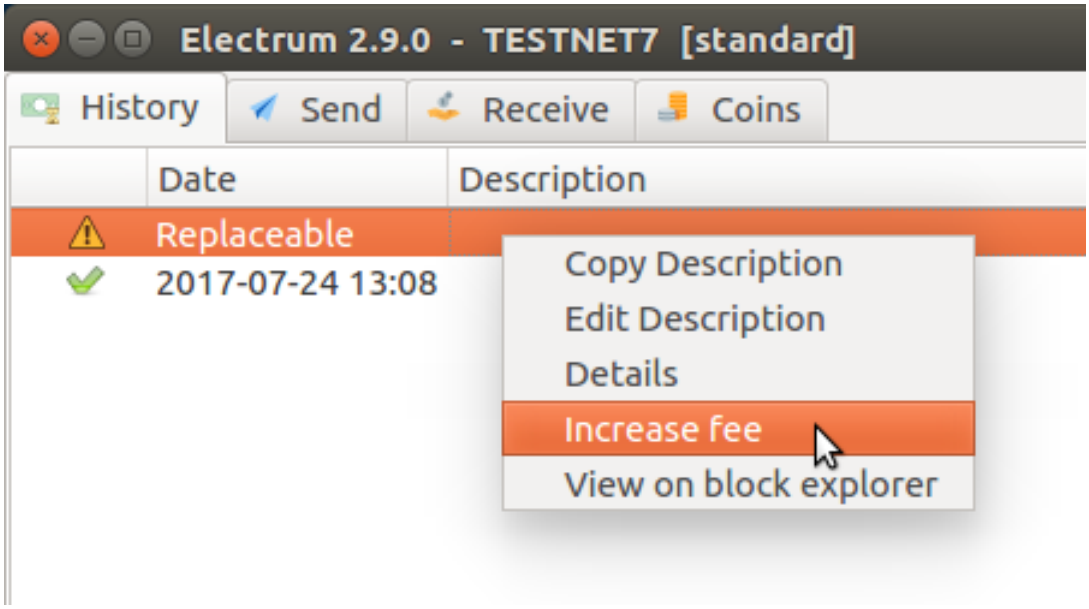
4. 1 つのネットワークにトランザクションが承認されるまで待つ

Âă a. ネットネットワークパネルを介してチェーンを切り替え、トランザクションステータスを監視する

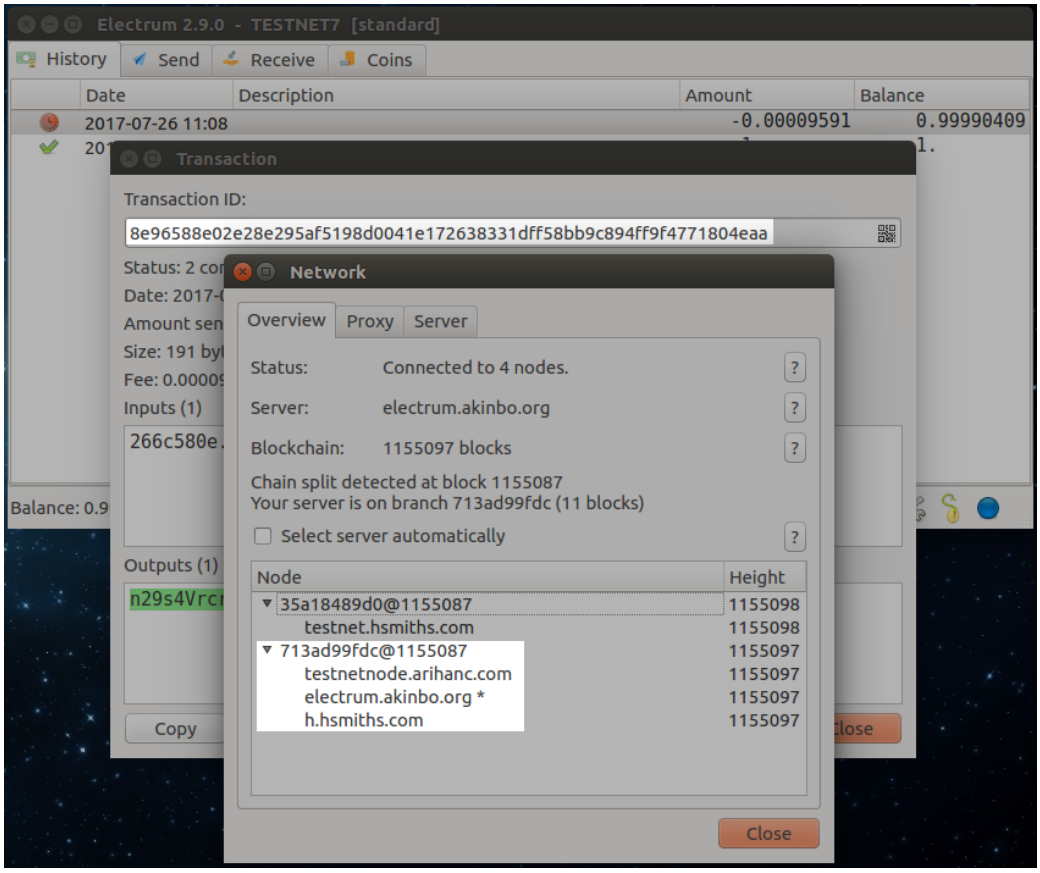
Âă b. 1 つのチェーン上でトランザクションが承認されるのを確認するまで待つ

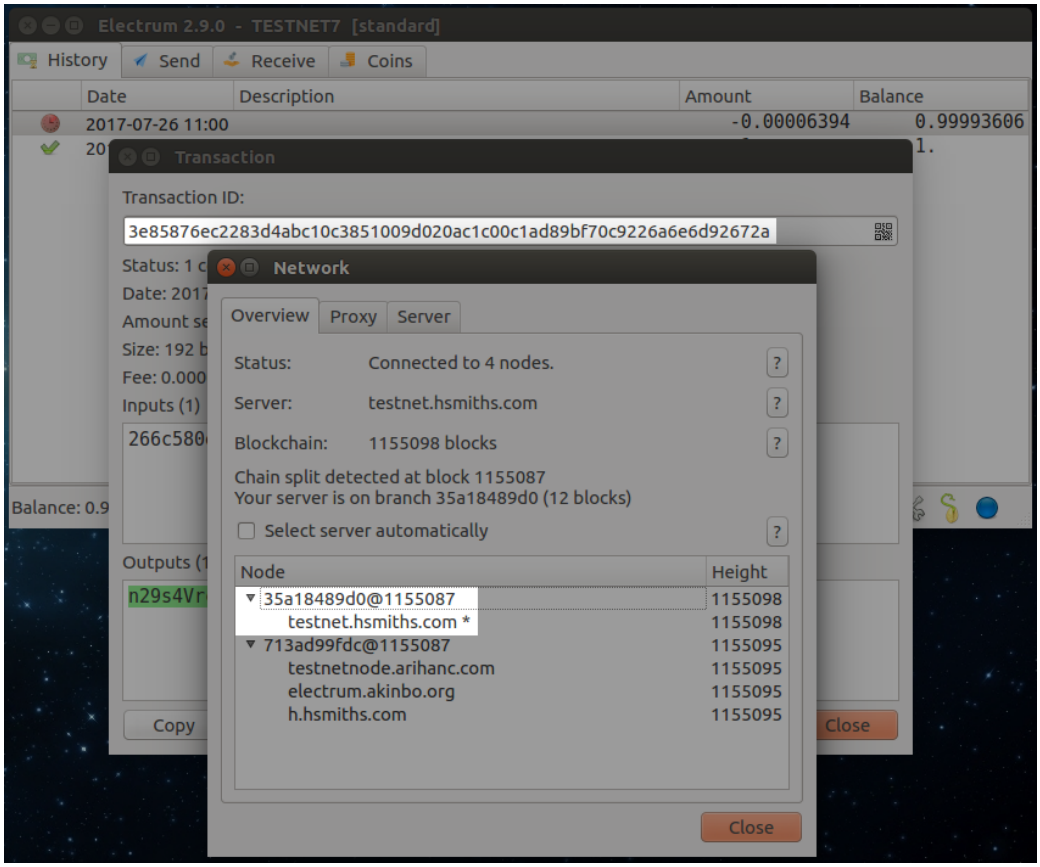


Âă c. 即座に未承認トランザクションで"RBF"を使用して"Increase fee"を行う



5. トランザクションが両方のチェーンで承認されるのを待つ
6. トランザクションがそれぞれのチェーンで異なる TXID を持っていることを確認する


Electrum 2.9.0 - TESTNET7 [standard]
History
Send
Receive
Coins
Date
Description
Amount
Balance
2017-07-26 11:08
-0.00009591
0.99990409
Transaction
Transaction ID:
8e96588e02e28e295af5198d0041e172638331dff58bb9c894ff9f4771804eaa
Network
Overview
Proxy
Server
Status:
Connected to 4 nodes.
Server:
electrum.akinbo.org
Blockchain:
1155097 blocks
Chain split detected at block 1155087
Your server is on branch 713ad99fdc (11 blocks)
Select server automatically
Node
Height
35a18489d0@1155087
1155098
testnet.hsmiths.com
1155098
713ad99fdc@1155087
1155097
testnetnode.arihanc.com
1155097
electrum.akinbo.org *
1155097
h.hsmiths.com
1155097
Close
Inputs (1)
Outputs (1)
Copy
Balance:



これでもうコインがそれぞれのチェーンで別々に使用できます。もし失敗しても、あなた自身に送信しているため害はありません。もう一度挑戦してください。

2.5 Tor を通して Electrum を使用する

Tor を通して Electrum を使用するには二つの主な方法があります。一つ目の方法は最も秘匿されますが接続するサーバを最も信用する必要もあります。なぜなら通常、Electrum はわずかな数のサーバーに接続してブロックヘッダをダウンロート、検証しているからです。これが悪意あるサーバがあなたに偽物の情報を送信することを難しくしています。しかしながら、これらのヘッダを求めて.onion サーバに接続することができるゆえにプライバシーの問題を発生させています。

このように二つの異なるオプションは、1 つのサーバだけに接続しブロックヘッダとトランザクション情報をサーバから取得する方法と、八つのブロックヘッダサーバに接続し一つの.onion サーバに一般用途のために接続する方法です。

2.5.1 アクティブな.onion サーバのリスト

ここのリストをチェックしてください

http://electrumserv.noip.me/onionservers.txt

このリストに追加したい場合次にメールを送ってください：

danielcryptos@gmail.com

2.5.2 LINUX

2.5.3 オプション 1: 単一サーバ

注意：追加プライバシーのためにはいくつかのセキュリティを犠牲にすることを理解してください。

https://electrum.org/#download をチェックします

Python のソースから download をつかみます

依存関係をインストールしていることを確認します

sudo apt-get install python-qt4 python-pip

ダウンロードした Electrum を展開します

tar -xvzf Electrum-2.*.*.tar.gz

展開した Electurm フォルダに移動して起動します；

./electrum -1 -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

クイック説明

-1 は一つのサーバだけに接続することを意味します

-s はサーバを定義します。あなたの望む.onion サーバに変更することができます（トップのリストをチェックしてください）

-p は Tor ネットワークに参加するために使用するプロキシサーバを決めます。一般的にこれは localhost ですが後のポートビットは異なる可能性があります。

実行しているシステムによってはポートビットを変更する必要があります。

現在のポートは；

Tor ブラウザバンドル： 9150

一般的な Tor(インストールされたもの)： 9050

2.5.4 オプション 2：複数のサーバー。ただし Tor がメイン

Electrum をスタートするコマンドまでは同じで、-1 を削除

./electrum -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

これで Electrum を起動し右下にある緑または赤のアイコンをクリックすることでサーバ情報を持ってくることができます。

「Auto」のチェックを外しボックスの中に入力してください；

electrums3lojbuj.onion

50001

下部の SOCKS5 をプロキシに選択し、次は

localhost

9150 or 9050

2.5.5 WINDOWS

2.5.6 オプション 1：単一サーバに接続

Electrum をメインダウンロードページからインストール https://electrum.org/#download

注意：追加プライバシーのためにはいくつかのセキュリティを犠牲にすることを理解してください。

Windows では、デスクトップに Electrum アイコンがあるしょう。C:Program Files (x86)Electrum に Electrum フォルダを見つけられない場合、コピーを作成するためにコピー＆ペーストしてください。

electrum.exe を右クリックしてショートカットを作成してください。ここにはショートカットを作成できないと言われるのでデスクトップに代わりを作成するに OK してください。

新しいショートカットまたは古いもののコピーを右クリックしてプロパティに進みます。トップバーのショートカットをクリック、target（リンク先）という名前のボックス内をクリックします。

それはすでにふきだし (speech bubble) のものと似た何かが書かれているはずです。もし違ったとしても一致させようとして変更しないでください。

我々の目的は最後のふきだしの後にビットを追加することです。スペースを入れて / を入力したらコピー＆ペーストします。

"C:Program Files (x86)Electrumelectrum.exe" -1 -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

変更を適用し OK したら、貴方が望むなら一般タブに戻って"electrum.exe - Shortcut"と書かれているものを Electrum - Tor 等に変更してよいでしょう。

もう一度適用、OK をクリックしてください。

これでこのショートカットから ELectrum を起動した場合、一つの Tor サーバだけを使用するようになります。

クイック説明

-1 は一つのサーバのみに接続することを意味します。

-s はサーバを定義します。あなたの望む.onion サーバに変更することができます（トップのリストをチェックしてください）

-p は Tor ネットワークに参加するために使用するプロキシサーバを決めます。一般的にこれは localhost ですが後のポートビットは異なる可能性があります。

実行しているシステムによってはポートビットを変更する必要があります。

現在のポートは；

Tor ブラウザバンドル： 9150

一般的な Tor(インストールされたもの)： 9050

2.5.7 オプション 2

Windows では、デスクトップに Electrum アイコンがあるしょう。C:Program Files (x86)Electrum に Electrum フォルダを見つけられない場合、コピーを作成するためにコピー＆ペーストしてください。

electrum.exe を右クリックしてショートカットを作成してください。ここにはショートカットを作成できないと言われるのでデスクトップに代わりを作成するに OK してください。

新しいショートカットまたは古いもののコピーを右クリックしてプロパティに進みます。トップバーのショートカットをクリック、target（リンク先）という名前のボックス内をクリックします。

それはすでにふきだし (speech bubble) のものと似た何かが書かれているはずです。もし違ったとしても一致させようとして変更しないでください。

我々の目的は最後のふきだしの後にビットを追加することです。スペースを入れて / を入力したらコピー＆ペーストします。

"C:Program Files (x86)Electrumelectrum.exe" -s electrums3lojbuj.onion:50001:t -p socks5:localhost:9050

変更を適用し OK したら、貴方が望むなら一般タブに戻って"electrum.exe - Shortcut"と書かれているものを Electrum - Tor 等に変更してよいでしょう。

もう一度適用、OK をクリックしてください。

これでこのショートカットから ELectrum を起動した場合、一つの Tor サーバだけを使用するようになります。

実行しているシステムによってはポートビットを変更する必要があります。

現在のポートは；

Tor ブラウザバンドル： 9150

一般的な Tor(インストールされたもの)： 9050

これで Electrum を起動し右下にある緑または赤のアイコンをクリックすることでサーバ情報を持ってくることができます。「Auto」のチェックを外しボックスの中に入力してください；

electrums3lojbuj.onion

50001

下部の SOCKS5 をプロキシに選択し、次は

localhost

9150 or 9050

2.6 Verifying GPG signature of Electrum using Linux command line

This can be used to verify the authenticity of Electrum binaries/sources.

Download only from electrum.org and remember to check the gpg signature again every time you download a new version

2.6.1 Obtain public GPG key for ThomasV

In a terminal enter (or copy):

```
gpg --keyserver keys.gnupg.net --recv-keys 6694D8DE7BE8EE5631BED9502BD5824B7F9470E6
```

You should be able to substitute any public GPG keyserver if keys.gnupg.net is (temporarily) not working

2.6.2 Download Electrum and signature file (asc)

Download the Python Electrum-<version>.tar.gz or AppImage file

Right click on the signature file and save it as well

2.6.3 Verify GPG signature

Run the following command from the same directory you saved the files replacing <electrum file> with the one actually downloaded:

```
gpg --verify <electrum file>.asc <electrum file>
```

The message should say:

```
Good signature from "Thomas Voegtlin (https://electrum.org) <thomasv@electrum.org>
```

and

```
Primary key fingerprint: 6694 D8DE 7BE8 EE56 31BE  D950 2BD5 824B 7F94 70E6
```

You can ignore this:

```
WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
```

as it simply means you have not established a web of trust with other GPG users

第 3 章

For developers

3.1 Python コンソール

ほとんどの ELectrum コマンドはコマンドラインだけでなく GUI Python コンソールでも使用できます。

わかりやすくするために JSON で表示されることがありますが結果は Python オブジェクトです。

listunspent() を呼び出して、ウォレット内の未使用アウトプットの一覧を確認しましょう。

```
>> listunspent()
[
 {
     "address": "12cmY5RHRgx8KkUKASDcDYRotget9FNso3",
     "index": 0,
     "raw_output_script": "76a91411bbdc6e3a27c44644d83f783ca7df3bdc2778e688ac",
     "tx_hash": "e7029df9ac8735b04e8e957d0ce73987b5c9c5e920ec4a445130cdeca654f096",
     "value": 0.01
 },
 {
     "address": "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF",
     "index": 0,
     "raw_output_script": "76a914aaf437e25805f288141bfcdc27887ee5492bd13188ac",
     "tx_hash": "b30edf57ca2a31560b5b6e8dfe567734eb9f7d3259bb334653276efe520735df",
     "value": 9.04735316
 }
]
```

結果は JSON として表示されることに注意してください。しかし、Python 変数に保存すると、Python オブジェクトとして表示されます。

```
>> u = listunspent()
>> u
[{'tx_hash': u'e7029df9ac8735b04e8e957d0ce73987b5c9c5e920ec4a445130cdeca654f096',
 'index': 0, 'raw_output_script': '76a91411bbdc6e3a27c44644d83f783ca7df3bdc2778e688ac',
 'value': 0.01, 'address': '12cmY5RHRgx8KkUKASDcDYRotget9FNso3'}, {'tx_hash':
↪'b30edf57ca2a31560b5b6e8dfe567734eb9f7d3259bb334653276efe520735df', 'index': 0, 'raw_
↪output_script': '76a914aaf437e25805f288141bfcdc27887ee5492bd13188ac', 'value': 9.
↪04735316, 'address': '1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF'}]
```

(次のページに続く)

(前のページからの続き)

これにより、Electrum コマンドと Python を組み合わせることができます。たとえば、先ほどの結果中のアドレスのみを選択します。

```
>> map(lambda x:x.get('address'), listunspent())
[
 "12cmY5RHRgx8KkUKASDcDYRotget9FNso3",
 "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF"
]
```

ここでは、未使用アウトプットを持つすべてのアドレスの秘密鍵をダンプするために、listunspent と dumpprivkeys の 2 つのコマンドを組み合わせています:

```
>> dumpprivkeys( map(lambda x:x.get('address'), listunspent()) )
{
 "12cmY5RHRgx8KkUKASDcDYRotget9FNso3":
↪"**************************************************",
 "1GavSCND6TB7HuCnJSTEbHEmCctNGeJwXF":
↪"**************************************************"
}
```

ウォレットが暗号化されている場合、dumpprivkey にはパスワードが必要です。GUI メソッドは gui 変数から利用できます。たとえば、gui.show_qrcode を使用して文字列から QR コードを表示することができます。例：

```
gui.show_qrcode(dumpprivkey(listunspent()[0]['address']))
```

3.2 簡易支払い検証

Simple Payment Verification（SPV）は、ナカモトサトシの論文に記載されている手法です。SPV によって、ブロックチェーン全体をダウンロードすることなく、Monacoin ブロックチェーンにトランザクションが含まれていることを確認できる軽量クライアントができます。SPV クライアントでは、完全なブロックよりはるかにサイズが小さいブロックヘッダをダウンロードするだけで済みます。トランザクションがブロック内に含まれていることを確認するために、SPV クライアントはマークルブランチの形式で内容の証明を要求します。

SPV クライアントは、送信された情報とサーバを信用する必要がないため、Web ウォレットよりも高いセキュリティを提供します。

参照： Bitcoin: A peer-to-peer Electronic Cash System

3.3 Electrum Seed バージョンシステム

このドキュメントでは Electrum（バージョン 2.0 以上）で用いられている Seed バージョンシステムのことを説明しています。

3.3.1 説明

Electrum は自然言語で出来た Seed フレーズから秘密鍵とアドレスを導き出しています。バージョン 2.0 以降、Electrum Seed フレーズはバージョン番号を含み、その目的は秘密鍵とアドレスを導き出すためにはどのデリベーションをたどるべきかを示すことです。

固定単語リストへの依存を排除するために、マスター秘密鍵およびバージョン番号は両方とも、UTF8 正規化された Seed フレーズのハッシュによって得られます。バージョン番号は次の最初のビットから得られます。

```
hmac_sha_512("Seed version", seed_phrase)
```

バージョン番号は Seed の整合性を確かめるためにも使用されます。間違いのないように、Seed フレーズは登録済のバージョン番号を提示しなければなりません。

3.3.2 動機

旧バージョンの Electrum(2.0 より前) は Seed フレーズとエントロピーの双方向エンコーディングを用いていました。このタイプのエンコーディングは固定の単語リストを必要とします。これが意味するのは将来のバージョンの Electrum が過去の Seed フレーズの解読を可能にするために全く同じ単語リストを備えなければならないということです。

BIP39 は Electrum の二年後に発表されました。BIP39 の Seed はチェックサムを含み、タイプミスを検出するのを補助しています。しかしながら BIP39 にはかつての Electrum Seed フレーズと同じ欠点があります：

- 固定の単語リストが依然として要求されます。我々の勧告を受けて、BIP39 の著者は鍵とアドレスを生成するのに単語リストには依存しない方法をとりました。しかしながら、BIP39 はチェックサムを計算するためにその単語リストを必要とします。チェックサムには明らかに矛盾があり、我々の勧告の目的を台無しにさせています。この問題は BIP39 が言語ごとに一つの単語リストを作成しようと提案したことでさらに悪化しました。これが BIP39 Seed フレーズの移植性を脅かしています。
- BIP39 Seed フレーズはバージョン番号を含みません。つまりソフトウェアは常に鍵とアドレスを生成する方法を知っていなければなりません。BIP43 ではウォレットソフトウェアは BIP32 フレームワーク内に存在する様々な生成スキームを試行するようにすることを提案しています。これは著しく非効率的であり、今後のウォレットがそれまでに利用された生成メソッドのすべてをサポートするという仮定に基づいています。もし将来、ウォレットの開発者が特定の生成メソッドは廃止予定だからと、その実行をやめることに決めた場合、そのソフトウェアは、サポートしていない該当 Seed フレーズを検出することができず、代わりに空のウォレットを返すでしょう。これはユーザーの資産を脅かします。

これらの理由から、Electrum は BIP39 Seed の生成は行ないません。バージョン 2.0 から、Electrum は以下の Seed バージョンシステムを使用してこれらの問題に対処しています。

Electrum2.0 は UTF8 正規化された固定の単語リストに依存しない Seed フレーズのハッシュから鍵とアドレスを生成します。つまり単語リストは Seed は移植性を保ったままウォレット間で異なることができます。そして今日生成された Seed をデコードするこために、将来のウォレットの実装では今日の単語リストを必要としません。これによって前方互換性のコストを減らしています。

3.3.3 バージョン番号

バージョン番号は Seed フレーズから生成されたハッシュのプレフィックスです。長さは 4bit の倍数で、以下のように計算されます：

```
def version_number(seed_phrase):
  # normalize seed
  normalized = prepare_seed(seed_phrase)
  # compute hash
  h = hmac_sha_512("Seed version", normalized)
  # use hex encoding, because prefix length is a multiple of 4 bits
  s = h.encode('hex')
  # the length of the prefix is written on the fist 4 bits
  # for example, the prefix '101' is of length 4*3 bits = 4*(1+2)
  length = int(s[0]) + 2
  # read the prefix
  prefix = s[0:length]
  # return version number
  return hex(int(prefix, 16))
```

正規化関数 (prepare_seed) は単語の間のスペース 1 つを除いてすべて削除します。また発音区別記号、アジアの CJK 文字 (CJK characters) も削除されます。

3.3.4 登録済み番号のリスト

以下のバージョン番号が Electrum Seed に使われています。

Number	Type	Description
0x01	Standard	P2PKH and Multisig P2SH wallets
0x100	Segwit	Segwit: P2WPKH and P2WSH wallets
0x101	2FA	Two-factor authenticated wallets

加えて、マスター公開鍵/秘密鍵のバージョンバイトどのタイプのアウトプットスクリプトが使用されるべきかを示しています。以下のプレフィックスがマスター公開鍵に使用されています。 here.

3.3.5 Seed 生成

Seed 生成において Seed フレーズがハッシュ化されると、結果のハッシュ値は正しいバージョン番号プレフィックスで始まらなければいけません。これは望むバージョン番号が作成されるまで nonce を挙げて Seed フレーズをハッシュ化しなおすことで達成されます。この要求は Seed のセキュリティを下げることはありません。（秘密鍵を生成するのに要求されるキーストレッチングのコスト次第）

3.3.6 セキュリティ推測

Electrum は現在は BIP39(2048 単語) と同じ単語リストを使用しています。代表的な Seed は 12 単語を有しており、Seed を選ぶ際には 132bit のエントロピーとなるようにします。

BIP39 に従って、キーストレッチングの 2048 ループがマスター秘密鍵の生成には追加されます。ハッシュに関して、これは Seed のセキュリティにさらなる 11bit を追加することと同等です（2048=2^11）。

攻撃者の観点からすると、Seed バージョンハッシュのプレフィックスを課すことで追加された制約は Seed のエントロピーを減少させることはありません。なぜなら Seed フレーズから得られる情報はないからです。攻撃者は 2^n の候補 Seed フレーズを列挙する必要があり、n は Seed を生成するのに使われたエントロピーの bit 数です。

しかしながら、攻撃者によって作成されたテストは候補 Seed が有効な Seed でない場合即座に返ってきます。なぜなら攻撃者は鍵を生成する必要がないからです。つまり付与されたプレフィックスはキーストレッチングの長さを削減しているということです。

n は Seed のエントロピー bit の数を示しており、m はキーストレッチングによって追加された難易度の bit 数を示しています。m=log2(stretching_iterations)。k はビットの中のプレフィックスの長さを示しています。

攻撃の各反復における、有効な Seed を得る確率 p = 2^-k

候補 Seed をテストするのに求められるハッシュの数 p * (1+2^m) + (1-p)*1 = 1 + 2^(m-k)

ゆえに、攻撃コストは 2^n * (1 + 2^(m-k))

これは 2^(n + m - k)、または 2^n と近似することができます。

Electrum に現在使用されている標準値は 2^(132 + 11 - 8) = 2^135 です。つまり標準の Electrum Seed はハッシュに関して 135bit エントロピーに相当するということです。

3.4 Electrum のプロトコル仕様

(NOTE: このドキュメントは古くなっています。プロトコルの 0.10 より新しいバージョンの場合はこちらを見てください https://electrumx.readthedocs.io/en/latest/protocol.html)

Stratum は Monacoin クライアントである Electrum とマイナーに主に使用されている一般的な Monacoin 通信プロトコルです。フォーマット————

Stratum プロトコルは JSON-RPC 2.0 に基づいています。（ただし、メッセージに「jsonrpc」情報は含まれていません）各メッセージは、終端文字（n）で終わらなければなりません。

3.4.1 リクエスト

Typical request looks like this:

典型的なリクエストは次のようになります。

```
{ "id": 0, "method":"some.stratum.method", "params": [] }
```

- id は 0 から始まり、すべてのメッセージは固有の ID 番号を持つ
- 可能なメソッドのリストと説明は以下のとおり
- params は配列。例：["1myfirstaddress"、 "1mysecondaddress"、 "1andonemoreaddress"]

応答 ‘‘‘ 応答は似ています：

```
{ "id": 0, "result": "616be06545e5dd7daec52338858b6674d29ee6234ff1d50120f060f79630543c
↪"}
```

- id はリクエストメッセージからコピーされます（このようにしてクライアントは各応答とリクエストをペアにすることができます）
- result can be:
  - 結果は次のようになります
    - null

  Âă - 文字列（上記のように）Âă - ハッシュ　例：

  ```
  { "nonce": 1122273605, "timestamp": 1407651121, "version": 2, "bits":␣
  ↪406305378 }
  ```

  Âă - ハッシュの配列　例：

  ```
  [ { "tx_hash:
  "b87bc42725143f37558a0b41a664786d9e991ba89d43a53844ed7b3752545d4f",
  "height": 314847 }, { "tx_hash":
  "616be06545e5dd7daec52338858b6674d29ee6234ff1d50120f060f79630543c",
  "height": 314853 } ]
  ```

メソッド

3.4.2 server.version

これは通常、クライアントの最初のメッセージであるきーっぷアライブメッセージとして毎分送信されます。クライアントは自身のバージョンとサポートするプロトコルのバージョンを送信します。サーバはサポートするバージョンのプロトコルで応答します（通常サーバ側の数値が高いほど互換性があります）。

このドキュメントで説明されているプロトコルのバージョンは 0.10 です。

request:

```
{ "id": 0, "method": "server.version", "params": [ "1.9.5", "0.6" ] }
```

response:

```
{ "id": 0, "result": "0.8" }
```

3.4.3 server.banner

request:

```
{ "id": 1, "method": "server.banner", "params": [] }
```

3.4.4 server.donation_address

3.4.5 server.peers.subscribe

クライアントは、このやり方で他のアクティブなサーバのリストを要求することができます。サーバは IRC チャネル（#electrum at freenode.net）に接続されており、互いに見ることができます。各サーバはそのバージョン、各アドレスの履歴プルーンリミット（"p100", "p10000" 等。数字はそれぞれのアドレスのためにいくつのトランザクションをサーバが保存しておくかを意味している）、サポートプロトコル（"t" = tcp@50001, "h" = http@8081, "s" = tcp/tls@50002, "g" = https@8082; 非標準のポートはこのように告知される: "t3300" for tcp on port 3300）を告知します。

注意 執筆時点ではこのメソッドの真のサブスクリプション実装は存在せず、サーバは応答を一度だけ送信します。サーバは今のところ通知の送信をしません。

request:

```
{ "id": 3, "method":
"server.peers.subscribe", "params": [] }<br/>
```

response:

```
{ "id": 3, "result": [ [ "83.212.111.114",
"electrum.stepkrav.pw", [ "v0.9", "p100", "t", "h", "s",
"g" ] ], [ "23.94.27.149", "ultra-feather.net", [ "v0.9",
"p10000", "t", "h", "s", "g" ] ], [ "88.198.241.196",
"electrum.be", [ "v0.9", "p10000", "t", "h", "s", "g" ] ] ]
}
```

3.4.6 blockchain.numblocks.subscribe

新たなブロックの高さに関する通知をクライアントに送信するリクエスト。現在のブロック高を返答します。

request:

```
{ "id": 5, "method":
"blockchain.numblocks.subscribe", "params": [] }
```

response:

```
{ "id": 5, "result": 316024 }
```

message:

```
{ "id": null, "method":
"blockchain.numblocks.subscribe", "params": 316024 }
```

3.4.7 blockchain.headers.subscribe

解析したブロックヘッダの形式で新たなブロックについての通知をクライアントに送信するリクエスト。

request:

```
{ "id": 5, "method":
"blockchain.headers.subscribe", "params": [] }
```

response:

```
{ "id": 5, "result": { "nonce":
3355909169, "prev_block_hash":
"0000000000000002b3ef284c2c754ab6e6abc40a0e31a974f966d8a2b4d5206",
"timestamp": 1408252887, "merkle_root":
"6d979a3d8d0f8757ed96adcd4781b9707cc192824e398679833abcb2afdf8d73",
"block_height": 316023, "utxo_root":
```

(次のページに続く)

(前のページからの続き)

```
"4220a1a3ed99d2621c397c742e81c95be054c81078d7eeb34736e2cdd7506a03",
"version": 2, "bits": 406305378 } }
```

message:

```
{ "id": null, "method":
"blockchain.headers.subscribe", "params": [ { "nonce":
881881510, "prev_block_hash":
"0000000000000001ba892b1717690900ae476857120a78fb50825f8b67a42d4",
"timestamp": 1408255430, "merkle_root":
"8e92bdbf1c5c581b5942fc290c6c52c586f091b279ea79d4e21460e138023839",
"block_height": 316024, "utxo_root":
"060f780c0dd07c4289aaaa2ef24723f73380095b31d60795e1308170ec742ffb",
"version": 2, "bits": 406305378 } ] }
```

3.4.8 blockchain.address.subscribe

与えられたアドレスのステータス（例えばトランザクション履歴）が変化したときに通知をクライアントに送信するリクエスト。ステータスとはトランザクション履歴のハッシュのことです。アドレスにトランザクションがまだない場合、ステータスは null です。

request:

```
{ "id": 6, "method":"blockchain.address.subscribe", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{ "id": 6, "result":"b87bc42725143f37558a0b41a664786d9e991ba89d43a53844ed7b3752545d4f"␣
↪}
```

message:

```
{ "id": null, "method":"blockchain.address.subscribe", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L",
↪"690ce08a148447f482eb3a74d714f30a6d4fe06a918a0893d823fd4aca4df580"]}
```

3.4.9 blockchain.address.get_history

指定されたアドレスに対して、トランザクションのリストとその高さ（および新しいバージョンの手数料）が返されます。

request:

```
{"id": 1, "method": "blockchain.address.get_history", "params": [
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{"id": 1, "result": [{"tx_hash":
↪"ac9cd2f02ac3423b022e86708b66aa456a7c863b9730f7ce5bc24066031fdced", "height": 340235}
↪, {"tx_hash": "c4a86b1324f0a1217c80829e9209900bc1862beb23e618f1be4404145baa5ef3",
↪"height": 340237}]}
{"jsonrpc": "2.0", "id": 1, "result": [{"tx_hash":
↪"16c2976eccd2b6fc937d24a3a9f3477b88a18b2c0cdbe58c40ee774b5291a0fe", "height": 0, "fee
↪": 225}]}
```

3.4.10 blockchain.address.get_mempool

3.4.11 blockchain.address.get_balance

request:

```
{ "id": 1, "method":"blockchain.address.get_balance", "params":[
↪"1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }
```

response:

```
{"id": 1, "result": {"confirmed": 533506535, "unconfirmed": 27060000}}
```

3.4.12 blockchain.address.get_proof

3.4.13 blockchain.address.listunspent

request:

```
{ "id": 1, "method":
"blockchain.address.listunspent", "params":
["1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L"] }<br/>
```

response:

```
{"id": 1, "result": [{"tx_hash":
"561534ec392fa8eebf5779b233232f7f7df5fd5179c3c640d84378ee6274686b",
"tx_pos": 0, "value": 24990000, "height": 340242},
```

(次のページに続く)

(前のページからの続き)


```
{"tx_hash":"620238ab90af02713f3aef314f68c1d695bbc2e9652b38c31c025d58ec3ba968",
"tx_pos": 1, "value": 19890000, "height": 340242}]}
```


3.4.14 blockchain.utxo.get_address

3.4.15 blockchain.block.get_header

3.4.16 blockchain.block.get_chunk

3.4.17 blockchain.transaction.broadcast

生のトランザクション（シリアル化、16 進数エンコード済）をネットワークに送信します。トランザクション id を返すか、トランザクションが何かしらの理由で無効な場合はエラーを返します。

request:


```
{ "id": 1, "method":
"blockchain.transaction.broadcast", "params":

↪"0100000002f327e86da3e66bd20e1129b1fb36d07056f0b9a117199e759396526b8f3a207800000000000ffffffff0ede0
↪" }<br/>
```


response:


```
{"id": 1, "result": "561534ec392fa8eebf5779b233232f7f7df5fd5179c3c640d84378ee6274686b"}
```


3.4.18 blockchain.transaction.get_merkle

blockchain.transaction.get_merkle [$txid, $txHeight]

3.4.19 blockchain.transaction.get

与えられた txid の生のトランザクション（16 進数エンコード済）を入手するためのメソッド。トランザクションが存在しない場合、エラーが返されます。

request:


```
{ "id": 17, "method":"blockchain.transaction.get", "params": [
"0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098"
] }
```

response:

```
{ "id": 17, "result":
↪"0100000001000000000000000000000000000000000000000000000000000000000000000000000000ffffffff0704ffff001d0104
↪"}
```

error:

```
{ "id": 17, "error": "{ u'message': u'No information available about transaction', u
↪'code': -5 }" }
```

3.4.20 blockchain.estimatefee

特定のブロック数以内にトランザクションが取り込まれるために必要な kbyte あたりのトランザクション手数料を推定します。推定するのに十分な情報をノードが有していない場合、値-1 が返されます。

Parameter:トランザクションが取り込まれるまでに待つであろうブロック数

request:

```
{ "id": 17, "method": "blockchain.estimatefee", "params": [ 6 ] }
```

response:

```
{ "id": 17, "result": 0.00026809 }
{ "id": 17, "result": 1.169e-05 }
```

error:

```
{ "id": 17, "result": -1 }
```

外部リンク

- https://docs.google.com/a/palatinus.cz/document/d/17zHy1SUlhgtCMbypO8cHgpWH73V5iUQKk_0rWvMqSNs/edit?hl=en_US" Slush の Stratum プロトコルのオリジナル詳細
- http://mining.bitcoin.cz/stratum-mining

Stratum マイニング拡張の詳細

3.5 未署名、又は一部署名済みトランザクションのシリアライゼーション

Electrum2.0 はトランザクションのために拡張シリアライゼーション形式を使用します。この形式の目的は未署名、又は一部署名済みトランザクションを共同署名者又はコールドストレージに送信することです。

これは、トランザクションインプットの「pubkey」フィールドを拡張することによって実現されます。

3.5.1 拡張公開鍵

pubkey の最初のバイトを見ることで拡張公開鍵であるかどうかがわかります：

- 0x02, 0x03, 0x04：正当な Monacoin 公開鍵（圧縮されているかどうか）
- 0xFF, 0xFE, 0xFD：拡張公開鍵

拡張公開鍵には 3 種類あります：

- 0xFF: bip32 の拡張公開鍵とデリベーション
- 0xFE: Electrum のレガシーデリベーション：マスター公開鍵 + デリベーション
- 0xFD: 未知の公開鍵だが Monacoin のものであることはわかっている公開鍵

3.5.2 公開鍵

これが Monacoin における公開鍵の正しいシリアライゼーションです。

0x02 or 0x03	compressed public key (32 bytes)
0x04	uncompressed public key (64 bytes)

3.5.3 BIP32 デリベーション

0xFF	xpub (78 bytes)	bip32 derivation (2*k bytes)

3.5.4 Electrum のレガシーデリベーション

0xFE	mpk (64 bytes)	derivation (4 bytes)

3.5.5 Monacoin アドレス

アドレス（又はアウトプットスクリプトの hash 160）はわかるけど公開鍵がわからない場合に使用されます。共同署名者は公開鍵を知っている必要があります。

0xFD	hash_160_of_script (20 bytes)