پیوست ۱
گزارش معتبری از بیت‌کوین توسط ساتوشی ناکاموتو

این گزارش معتبر دقیقاً به طور کامل از روی همان نسخه‌ای که ساتوشی ناکاموتو در اکتبر ۲۰۰۸ منتشر کرد، در اینجا نسخه برداری شده است.

بیت‌کوین: یک سیستم ارز الکترونیکی نظیر به نظیر

ساتوشی ناکاموتو
satoshin@gmx.com
www.bitcoin.org

خلاصه

نسخه‌ی کاملاً نظیر به نظیری از ارز الکترونیکی، پرداخت‌های مستقیم آنلاین از یک طرف به طرف دیگر و بدون مراجعه به یک موسسه‌ی مالی را امکان‌پذیر می‌سازد. امضاهای دیجیتال بخشی از راهکار را ارائه می‌دهند، اما اگر برای جلوگیری از خرج دوباره هنوز هم طرف سوم مورد اعتماد مورد نیاز باشد، مزایای اصلی از بین می‌رود. ما با استفاده از یک شبکه‌ی نظیر به نظیر، راهکاری برای خرج مضاعف پیشنهاد می‌کنیم. شبکه با درهم‌سازی تراکنش‌ها در زنجیری از اثباتِ کار[1] بر آن‌ها مُهر زمانی می‌زند و فایلی تشکیل می‌دهد که بدون انجام دوباره‌ی اثبات کار، قابل تغییر نیست. طولانی‌ترین زنجیر، نه تنها به عنوان اثباتی بر توالی رویدادهای مشاهده‌شده عمل می‌کند، بلکه اثباتی است بر این‌که زنجیر مذکور از بزرگ‌ترین مخزن توان CPU حاصل می‌شود. تا زمانی که بیشتر توان پردازنده توسط گره‌هایی کنترل شود که برای هجوم به شبکه همکاری نمی‌کنند، آن گره‌ها طولانی‌ترین مهاجم‌های زنجیری و فرعی را ایجاد می‌کنند. خود شبکه به ساختاری کمینه نیاز دارد. پیام‌ها بر اساس بهترین

1. Proof of work

تلاش، منتشر می‌شوند و گره‌ها می‌توانند شبکه را به خواست خود ترک کنند و مجدداً به آن بپیوندند و طولانی‌ترین زنجیر اثباتِ کار را به عنوان اثبات آن‌چه اتفاق افتاده است، بپذیرند.

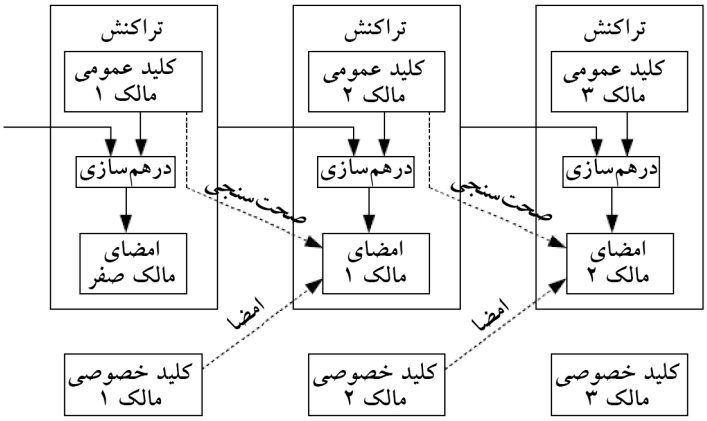
مقدمه

تجارت در اینترنت تقریباً به طور انحصاری بر موسسات مالی متکی است که به عنوان طرف سوم مورد اعتماد برای پردازش پرداخت‌های الکترونیکی فعالیت می‌کنند. در حالی که این سیستم برای بسیاری از تراکنش‌ها به اندازه‌ی کافی خوب عمل می‌کند، همچنان دارای نقاط ضعف ذاتی مدل مبتنی بر اعتماد است. تراکنش‌های کاملاً برگشت‌ناپذیر، واقعاً امکان‌پذیر نیستند، زیرا موسسات مالی نمی‌توانند از وساطت در اختلافات پرهیز کنند. هزینه‌ی وساطت، هزینه‌های تراکنش را افزایش می‌دهد، حداقل اندازه‌ی تراکنش‌های عملی را محدود می‌کند و امکان انجام تراکنش‌های کوچک موردی را نیز منتفی می‌کند. همچنین، با از دست رفتن توانایی انجام پرداخت‌های برگشت‌ناپذیر برای سرویس‌های برگشت‌ناپذیر، هزینه‌های بیشتری تحمیل می‌شود. با امکان برگشت‌پذیری، نیاز به اعتماد گسترش می‌یابد. فروشنده باید نسبت به مشتریان خود محتاط باشد و برای دریافت اطلاعات بیشتری از آن‌ها، نسبت به حالتی که در غیر این صورت نیاز داشت، بیشتر مزاحم آن‌ها می‌شود. اجتناب‌ناپذیر بودن درصد مشخصی از کلاهبرداری پذیرفته می‌شود. از این هزینه‌ها و عدم قطعیت‌های پرداخت به صورت حضوری می‌توان با استفاده از وجه رایج فیزیکی جلوگیری کرد، اما هیچ راهکاری برای پرداخت از طریق کانال ارتباطی بدون یک طرف سوم قابل اعتماد وجود ندارد.

آن‌چه لازم است، یک سیستم پرداخت الکترونیکی مبتنی بر اثبات رمزنگاری به جای اعتماد است و به هر دو طرف امکان می‌دهد بدون نیاز مستقیم به طرف سوم قابل اعتماد با یکدیگر ارتباط برقرار کنند. تراکنش‌هایی که برگشت‌پذیری آن‌ها از نظر محاسباتی غیرعملی است، از فروشندگان در برابر کلاهبرداری محافظت می‌کنند و راهکارهای معمول سپردن پول را به راحتی می‌توان برای محافظت از خریداران پیاده‌سازی کرد. در این مقاله، راهکاری برای مشکل خرج دوباره با استفاده از یک سرور مُهرِ زمانی توزیع‌شده‌ی نظیر به نظیر برای تولید اثبات محاسباتی ترتیب زمانی تراکنش‌ها انجام می‌دهیم. این سیستم تا زمانی اَمن است که گره‌های درستکار، بیش از هر گروه گره‌های مهاجم همکار، توان پردازنده بیشتری را کنترل کنند.

تراکنش‌ها

یک سکه‌ی الکترونیکی به عنوان زنجیری از امضاهای دیجیتال تعریف می‌شود. هر مالک با امضای دیجیتالی درهم‌سازی تراکنش قبلی و کلید عمومی مالک بعدی و افزودن این موارد به انتهای سکه، آن را به دیگری منتقل می‌کند. گیرنده می‌تواند امضاها را تأیید کند تا زنجیر مالکیت را تأیید کند.

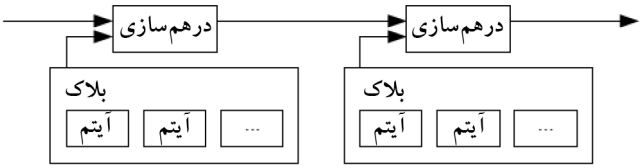


البته مشکل این است که گیرنده نمی‌تواند تأیید کند که یکی از مالکان سکه را دو بار خرج نکرده است. یک راهکار رایج، معرفی یک مقام مرکزی معتبر یا ضرابخانه است که هر تراکنش را از لحاظ خرج مضاعف بررسی کند. پس از هر تراکنش، سکه باید برای تهیه سکه جدید به ضرابخانه بازگردانده شود و فقط سکه‌های صادر شده‌ی مستقیم از ضرابخانه که دو بار خرج نشده باشند، مورد اعتماد خواهند بود. مشکل این راهکار آن است که سرنوشت کل سیستم پول بستگی به شرکتی دارد که ضرابخانه را اداره می‌کند و هر تراکنشی باید از طریق آن انجام شود، یعنی درست مانند بانک.

ما به روشی نیاز داریم تا گیرنده بداند صاحبان قبلی هیچ‌گونه تراکنش قبلی را امضا نکرده‌اند. برای اهداف ما، نخستین تراکنش، همان تراکنشی است که به حساب می‌آید، بنابراین ما به تلاش‌های بعدی برای خرج دوباره اهمیت نمی‌دهیم. تنها راه تأیید عدم وجود تراکنش آگاهی از کلیه تراکنش‌ها است. در مدل مبتنی بر ضرابخانه، ضرابخانه از همه‌ی تراکنش‌ها آگاه بود و تصمیم می‌گرفت که نخست کدام تراکنش برسد. برای تحقق این امر بدون داشتن یک طرف سوم قابل اعتماد، تراکنش‌ها باید به صورت علنی اعلام شوند [۱] و به سیستمی نیاز داریم تا شرکت‌کنندگان بر سر تاریخچه‌ی واحدی از ترتیب دریافت تراکنش‌ها به توافق برسند. گیرنده به این اثبات نیاز دارد که در زمان انجام هر تراکنش، اکثریت گره‌ها توافق کرده‌اند که آن تراکنش برای نخستین بار دریافت شده است.

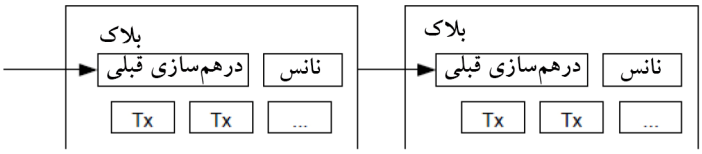
سرور مُهرِ زمانی

راه حل پیشنهادی ما با سرور مُهرِ زمانی آغاز می‌شود. سرور مهر زمانی با زدن مهر زمانی بر درهم‌سازی بلاکی از آیتم‌ها و انتشار گسترده‌ی درهم‌سازی، مانند یک روزنامه یا پست Usenet، عمل می‌کند [۵-۲]. بدیهی است مهر زمانی ثابت می‌کند که داده‌ها باید در زمان معین وجود داشته باشند تا مشمول درهم‌سازی شوند. هر مهر زمانی شامل مهر زمانی پیشین در درهم‌سازی بوده زنجیری را تشکیل می‌دهد و با افزوده شدن هر مُهر زمانی، قبلی‌ها بیشتر تقویت می‌شوند.



اثبات کار

برای پیاده‌سازی سرور مُهر زمانی توزیع شده مبتنی بر مدل نظیر به نظیر، به استفاده از سیستم اثبات کار مشابه با هَشکَش[1] آدام بِک[2] [۶] و نه روزنامه یا پست‌های Usenet نیاز داریم. اثبات کار شامل اسکن برای مقداری است که وقتی درهم‌سازی شد، برای مثال با SHA-256، درهم‌سازی با تعدادی بیت صفر آغاز می‌شود. میانگین کار مورد نیاز با تعداد بیت صفر مورد نیاز به طور نمایی رشد می‌کند و با اجرای یک درهم‌سازی واحد قابل صحت‌سنجی است. برای شبکه‌ی مهر زمانی، مادامی که مقداری پیدا نشود که به درهم‌سازی بلاک، بیت‌های صفر مورد نیاز را بدهد، اثبات کار را با افزایش یک نانس در بلاک پیاده‌سازی می‌کنیم. هنگامی که CPU برای به نتیجه رساندن اثبات کار تلاش کرد، بلاک بدون انجام دوباره‌ی کار قابل تغییر نخواهد بود. همان‌طور که بلاک‌های بعد از آن زنجیر می‌شوند، کار لازم برای تغییر بلاک شامل انجام دوباره‌ی همه بلاک‌های پس از آن خواهد شد.



اثبات کار همچنین مشکل تعیین نمایندگی در تصمیم‌گیری‌های اکثریتی را حل می‌کند. اگر اکثریت مبتنی بر "یک رأی به ازای هر آدرس IP" می‌بود، هر کسی که قادر به تخصیص تعداد زیادی از IPها بود، می‌توانست آن را واژگون سازد. اثبات کار اساساً مبتنی بر "یک رأی به ازای هر CPU" است. تصمیم اکثریت توسط طولانی‌ترین زنجیر نمایش داده می‌شود که بیشترین تلاش اثبات کار برای آن صرف شده است. اگر اکثر توان پردازنده توسط گره‌های درستکار کنترل شود، زنجیر درستکار سریع‌تر رشد می‌کند و از هر زنجیر رقیب، فراتر می‌رود. برای تغییر یک بلاک گذشته، مهاجم ناگزیر است اثبات کار بلاک و تمامی بلاک‌های پس از آن را دوباره انجام دهد و سپس خودش را به گره‌های درستکار برساند و از آن‌ها پیشی بگیرد. بعداً نشان خواهیم داد که با افزوده شدن بلاک‌های بعدی، احتمال حمله‌ی مهاجم کندتر به صورت نمایی کاهش می‌یابد.

برای جبران افزایش سرعت سخت‌افزار و تغییر علاقه به گره‌های در حال کار با گذر زمان، مشکل اثبات کار با میانگین متحرکی تعیین می‌شود که تعداد میانگین بلاک‌ها را در هر ساعت هدف قرار می‌دهد. اگر سرعت تولیدشان بیش از اندازه باشد، مشکل افزایش می‌یابد.

1. Hashcash 2. Adam Back

شبکه

مراحل اجرای شبکه به شرح زیر است:

۱. تراکنش‌های جدید در تمامی گره‌ها منتشر می‌شوند.
۲. هر گره تراکنش‌های جدید را درون یک بلاک جمع‌آوری می‌کند.
۳. هر گره برای یافتن یک اثبات کار دشوار برای بلاک خود کار می‌کند.
۴. هنگامی که گره‌ای اثبات کار را یافت، بلاک را برای تمامی گره‌ها منتشر می‌کند.
۵. گره‌ها بلاک را تنها در صورتی قبول می‌کنند که کلیه تراکنش‌های موجود در آن معتبر بوده قبلاً هزینه نشده باشد.
۶. گره‌ها با کار بر روی ایجاد بلاک بعدی در زنجیر، با استفاده از درهم‌سازی بلاک پذیرفته شده به عنوان درهم‌سازی قبلی، پذیرش بلاک را اعلام می‌کنند.

گره‌ها همواره طولانی‌ترین زنجیر را صحیح می‌دانند و به تلاش برای گسترش آن ادامه می‌دهند. اگر دو گره، همزمان نسخه‌های متفاوتی از بلاک بعدی را منتشر کنند، ممکن است برخی از گره‌ها یکی یا دیگری را دریافت کنند. در آن حالت، گره‌های مذکور روی نخستین موردی که دریافت کرده‌اند کار می‌کنند، اما در صورت طولانی‌ترشدن، شاخه‌ی دیگر را نیز نگه می‌دارند. این پیوند هنگامی شکسته خواهد شد که اثبات بعدی کار پیدا شود و یک شاخه طولانی‌تر شود. در آن صورت، گره‌هایی که روی شاخه‌ی دیگر کار می‌کردند، به شاخه‌ی دیگر تغییر وضعیت خواهد داد.

ضرورتی ندارد که انتشار تراکنش‌های جدید به همه‌ی گره‌ها برسد. مادامی که تراکنش‌ها به تعداد زیادی از گره‌ها برسند، پس از مدت زمانی کوتاه وارد یک بلاک می‌شوند. انتشار بلاک‌ها نسبت به پیام‌های جاافتاده نیز سختگیری چندانی ندارد. اگر یک گره، بلاکی را دریافت نکند، هنگام دریافت بلاک بعدی آن را درخواست می‌کند و متوجه می‌شود یکی را از دست داده است.

مشوق

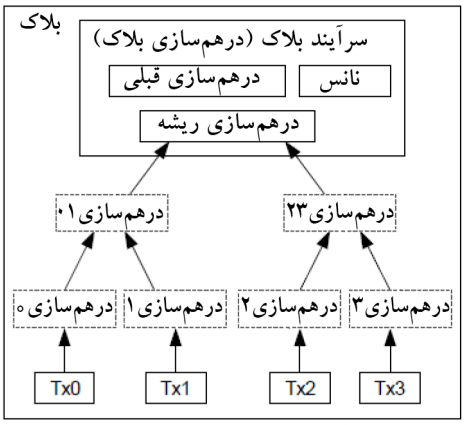
طبق قرارداد، نخستین تراکنش در یک بلاک، تراکنشی ویژه است که یک سکه‌ی جدید متعلق به ایجادکننده‌ی آن بلاک را آغاز می‌کند. این مشوقی برای پشتیبانی گره‌ها از شبکه راهی برای توزیع اولیه‌ی سکه‌ها در مدار فراهم می‌کند، زیرا مرجع متمرکزی برای صدور آن‌ها وجود ندارد. افزایش مداوم مبلغ سکه‌های جدید شبیه به این است که کارگران معدن طلا برای اضافه کردن طلای در گردش، منابع خود را خرج می‌کنند. در مورد ما، زمان CPU و برق منابعی هستند که مصرف می‌شوند.

هزینه‌ی مشوق ممکن است از کارمزد تراکنش‌ها نیز تأمین شود. اگر مقدار خروجی یک تراکنش کمتر از مقدار ورودی آن باشد، این اختلاف، برابر با کارمزد تراکنش است که به ارزش تشویقی بلاک حاوی تراکنش افزوده می‌شود. هنگامی که تعداد سکه‌های از پیش تعیین شده وارد گردش شود، مشوق می‌تواند کاملاً به کارمزدهای تراکنش منتقل شود و کاملاً عاری از تورم باشد.

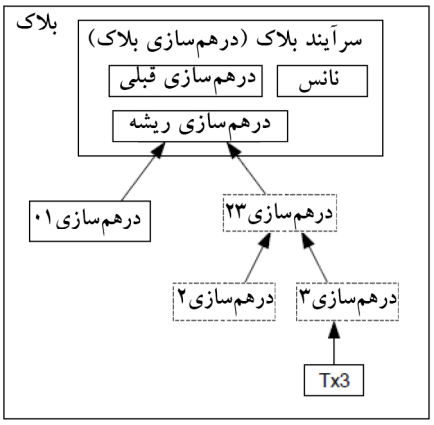
مشوق ممکن است به ترغیب گره‌ها برای درستکار ماندن کمک کند. اگر یک مهاجم حریص بتواند نسبت به همه‌ی گره‌های درستکار، توان CPU بیشتری را جمع کند، ناگزیر خواهد بود تا از میان استفاده از آن برای فریب دادن مردم با پرداخت‌های خودش یا استفاده از آن برای تولید سکه‌های جدید یکی را برگزیند. او باید در بازی با این قواعد (قواعدی که در مقایسه با مجموع افراد دیگر، سکه‌های جدید بیشتری به او می‌بخشد) نسبت به تضعیف سیستم و اعتبار ثروت خویش سود بیشتری ببیند.

بازیابی فضای دیسک

هنگامی که آخرین تراکنش در یک سکه، زیر بلاک‌های کافی دفن شد، تراکنش‌های خرج شده‌ی پیش از آن را می‌توان دور انداخت تا در فضای دیسک صرفه‌جویی شود. برای تسهیل در این امر بدون شکستن درهم‌سازی بلاک، تراکنش‌ها در یک درخت مرکل [۷] [۲] [۵] درهم‌سازی می‌شوند، در حالی که فقط ریشه‌ی این درخت در درهم‌سازی بلاک لحاظ می‌شود. بلاک‌های قدیمی را می‌توان با هرس کردن شاخه‌های درخت فشرده کرد. درهم‌سازی‌های داخلی نیازی به ذخیره‌سازی ندارند.



تراکنش‌های درهم‌سازی‌شده در درخت مرکل



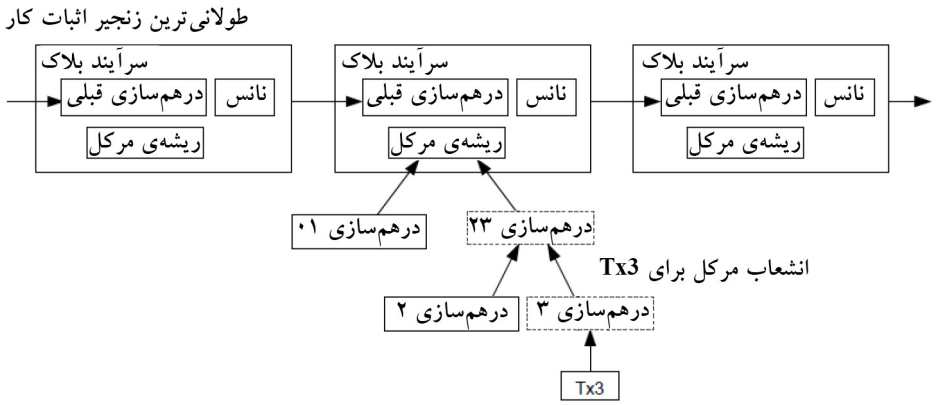
پس از هرس **Tx0-2** از بلاک

یک بلاک سرآیند بدون تراکنش، نزدیک به ۸۰ بایت طول دارد. اگر فرض کنیم هر ۱۰ دقیقه، یک بلاک جدید ساخته شود، حافظه‌ی لازم برای یک سال، 80 bytes * 6 * 24 * 365 = 4.2 MB می‌شود. با سیستم‌های کامپیوتری که از سال ۲۰۰۸ معمولاً با 2 گیگابایت رم به فروش می‌رسند، و از آنجا که بنا به قانون مور، رشد فعلی 1.2 گیگابایت در سال برای حافظه‌ی دستگاه‌ها پیش بینی می‌شود، حتی اگر ناگزیر از نگهداری بلاک هدرها در حافظه باشیم، فضای حافظه مشکلی ایجاد نخواهد کرد.

صحت‌سنجی پرداخت ساده

صحت‌سنجی پرداخت بدون اجرای گره‌ی شبکه‌ی کامل امکان‌پذیر است. کاربر فقط باید یک کپی از بلاک سرآیندهای طولانی‌ترین زنجیر اثبات کار را نگه دارد که این کپی را می‌تواند با درخواست از گره‌های شبکه (تا زمانی که متقاعد شود طولانی‌ترین زنجیر را دارد و شاخه‌ی مرکلی را به دست آورد

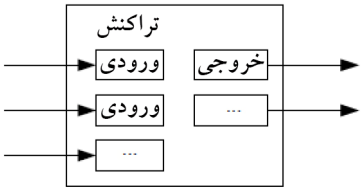
که تراکنش را به بلاکی پیوند می‌دهد که مهر زمانی آن را بر خود دارد) از آن خود کند. او نمی‌تواند تراکنش را برای خود بررسی کند، اما با پیوند دادن آن به مکانی در زنجیر، می‌تواند ببیند که یک گره‌ی شبکه آن را پذیرفته است و بلاک‌های افزوده شده‌ی پس از تأیید بیشتر شبکه، آن را پذیرفته‌اند.



به این ترتیب، تا زمانی که گره‌های درستکار شبکه را کنترل کنند، این صحت‌سنجی قابل اطمینان است، اما اگر مهاجمی بر شبکه تسلط داشته باشد، آسیب‌پذیرتر است. درحالی که گره‌های شبکه می‌توانند تراکنش‌ها را برای خودشان صحت‌سنجی کنند، مادامی که مهاجم تسلط خویش بر شبکه را حفظ کند، می‌تواند این روش ساده‌سازی شده را با تراکنش‌های ساختگی فریب دهد. یکی از راهکارهای محافظت در برابر این تهدید، پذیرش هشدار از گره‌های شبکه هنگام شناسایی یک بلاک نامعتبر است و از نرم‌افزار کاربر خواسته می‌شود که بلاک کامل و تراکنش‌های هشدار داده شده را برای تأیید ناسازگاری دانلود کند. کسب‌وکارهایی که پرداخت مکرر دریافت می‌کنند، احتمالاً هنوز هم می‌خواهند برای امنیت مستقل‌تر و صحت‌سنجی سریع‌تر، گره‌های خاص خودشان را اجرا کنند.

تلفیق و تقسیم ارزش

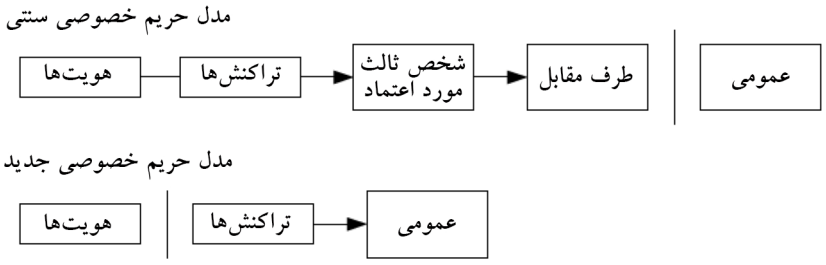
گرچه می‌توان سکه‌ها را جداگانه اداره کرد، انجام تراکنش جداگانه برای هر سنت از آن‌ها در یک انتقال، امری دشوار است. برای این‌که امکان تقسیم و ترکیب ارزش وجود داشته باشد، هر تراکنش حاوی چند ورودی و خروجی است. معمولاً یا یک چند ورودی واحد از یک تراکنش قبلی بزرگ‌تر یا چند ورودی که مقادیر کوچک‌تر را با هم تلفیق می‌کنند (با حداکثر دو خروجی) وجود دارد: یکی خروجی برای پرداخت و دیگری، در صورت وجود، بقیه‌ی پول را به فرستنده باز می‌گرداند.



لازم به ذکر است که گنجایش خروجی، در جایی که یک تراکنش به چند تراکنش بستگی داشته باشد و آن تراکنش‌ها نیز خود به بسیاری از تراکنش‌های دیگر بستگی داشته باشند، در این‌جا مشکلی ایجاد نمی‌کند. هرگز نیازی به استخراج یک کپی مستقل از تاریخچه‌ی تراکنش وجود ندارد.

حریم خصوصی

مدل بانکی سنتی با محدود کردن دسترسی به اطلاعات به طرف‌های درگیر و طرف سوم مورد اعتماد، به سطح حریم خصوصی می‌رسد. ضرورت اعلام عمومی کلیه‌ی تراکنش‌ها سد راه این روش می‌شود، اما با شکستن جریان اطلاعات در جای دیگر، می‌توان حریم خصوصی را حفظ کرد: یعنی با گمنام نگه داشتن کلیدهای عمومی. عموم مردم می‌توانند ببینند که شخصی مبلغی را برای شخص دیگری ارسال می‌کند، اما اطلاعاتی که این تراکنش را به کسی مربوط می‌کند، آشکار نخواهند شد. این شبیه به سطح اطلاعاتی است که از بورس اوراق بهادار منتشر می‌شود، یعنی جایی که زمان و اندازه‌ی تراکنش‌های فردی، علنی می‌شود، بی‌آن که گفته شود، طرفین چه کسانی بوده‌اند.



به عنوان یک دیوار آتش (فایروال) اضافی، برای هر تراکنش باید از یک جفت کلید جدید استفاده شود تا از اتصال آن‌ها به یک مالک مشترک جلوگیری شود. هنوز مقداری اتصال با تراکنش‌های چند ورودی پرهیزناپذیر است که الزاماً نشان می‌دهد ورودی‌های آن‌ها تنها به یک مالک تعلق دارد. خطری که وجود دارد این است که اگر هویت مالک یک کلید فاش شود، این اتصال می‌تواند تراکنش‌های دیگری را که متعلق به همان مالک است، فاش کند.

محاسبات

سناریوی مهاجمی را در تلاش برای تولید زنجیر دیگری سریع‌تر از زنجیر درستکار در نظر می‌گیریم. حتی اگر این منظور او محقق شود، سیستم را برای تغییرات دلخواه، مانند کره گرفتن از آب یا گرفتن پولی که هرگز متعلق به مهاجم نبوده است، باز نمی‌کند. قرار نیست گره‌ها تراکنش نامعتبر را به عنوان پرداخت بپذیرند و گره‌های درستکار هرگز بلاکی را که آن‌ها را شامل شود، نمی‌پذیرند. مهاجم فقط می‌تواند سعی کند یکی از تراکنش‌های خود را تغییر دهد تا بتواند پولی را که اخیراً خرج کرده، پس بگیرد.

مسابقه‌ی میان زنجیر درستکار و زنجیر مهاجم را می‌توان با یک گام تصادفی[۱] دوجمله‌ای مشخص کرد. رویداد موفقیت، زنجیر درستکاری است که به اندازه‌ی یک بلاک توسعه می‌یابد و آن را با +1 جلو می‌اندازد؛ رویداد شکست این است که زنجیر مهاجم یک به اندازه‌ی بلاک افزایش یابد و فاصله را با -1 کاهش دهد.

احتمال برخورداری یک مهاجم از کمبودی معین، مشابه مسأله‌ی پاکباختگی قمارباز است. فرض کنید قماربازی با اعتبار نامتناهی آغاز می‌کند و به طور بالقوه بی‌نهایت مرتبه بازی می‌کند تا به نقطه‌ی سربه‌سر برسد. احتمال دستیابی او به نقطه‌ی سربه‌سر، یا این‌که مهاجمی همیشه با زنجیر درستکار روبرو شود، می‌توانیم محاسبه کنیم [۸]:

p = احتمال این‌که یک گره درستکار، بلاک بعدی را بیابد.

q = احتمال این‌که مهاجم، بلاک بعدی را بیابد.

q_z = احتمال این‌که مهاجمی از z بلاک پشت سر، به بلاک موردنظر برسد.

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

با توجه به فرض ما که $p > q$ بود، با افزایش تعداد بلاک‌هایی که مهاجم باید پشت سر بگذارد تا به بلاک آخر برسد، این احتمال به طور نمایی کاهش می‌یابد. چنانچه بخت با او یار نباشد، اگر از همان ابتدا فرصتی مناسب سبب پرش ناگهانی او نشود، هرچه بیشتر عقب بماند، شانس موفقیت او با آهنگ تندتری کم می‌شود.

اکنون ببینیم گیرنده‌ی تراکنش جدید، پیش از آن‌که به اندازه‌ی کافی یقین حاصل کند فرستنده نمی‌تواند تراکنش را تغییر دهد، چقدر باید منتظر بماند. فرض کنیم فرستنده مهاجمی است که می‌خواهد گیرنده را قانع کند که مدتی به او پرداخت کرده است، سپس مدتی که گذشت، آن را به خودش بازپرداخت کند. هنگامی که این اتفاق رخ دهد، به گیرنده هشدار داده می‌شود، اما فرستنده امیدوار است بیش از اندازه دیر شود.

گیرنده یک جفت کلید جدید ایجاد می‌کند و کلید عمومی را کمی پیش از امضا به فرستنده می‌دهد. این مانع از آن می‌شود که ارسال‌کننده بتواند با کار کردن پیوسته روی زنجیری از بلاک‌ها، آن را زودهنگام آماده کند تا این‌که او به اندازه‌ی کافی بخت یار او باشد و به اندازه‌ی کافی جلو بیفتد و سپس در آن لحظه تراکنش را انجام دهد. پس از ارسال تراکنش، ارسال‌کننده‌ی متقلب آغاز به کار مخفیانه روی زنجیری موازی می‌کند که حاوی نسخه دیگری از تراکنش خود او است.

---

۱. Random Walk: ولگشت یا گام تصادفی یا گشت تصادفی، مطالعه‌ی رفتار یک مسیر تشکیل‌شده از گام‌های تصادفی و پی‌درپی با استفاده از ابزارهای ریاضی است.

گیرنده منتظر می‌ماند تا تراکنش به بلاک اضافه شود و z بلاک پس از آن پیوند یابند. او از پیشرفت دقیق مهاجم خبر ندارد، اما با فرض این‌که بلاک‌های درستکار میانگین زمان انتظار در هر بلاک را به خود اختصاص داده اند، پیشرفت بالقوه‌ی مهاجم، از توزیع پواسون با امید ریاضی زیر پیروی خواهد کرد:

$$\lambda = z\frac{q}{p}$$

برای به دست آوردن این که اکنون چقدر احتمال دارد تا مهاجم هنوز هم به انتهای زنجیر برسد، چگالی پواسون را برای هر میزان پیشرفتی که می‌تواند داشته باشد در احتمال رسیدن او از آن نقطه ضرب می‌کنیم:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

و با بازنویسی رابطه‌ی بالا برای پرهیز از جمع زدن بی‌نهایت توزیع داریم:

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!}(1-(q/p)^{(z-k)})$$

و باکدنویسی به زبان C:

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
   double p == 1.0 - q;
   double lambda == z * (q / p);
   double sum == 1.0;
   int i, k;
   for (k == 0; k <== z; k++)
   {
      double poisson == exp(-lambda);
      for (i == 1; i <== k; i++)
         poisson *== lambda / i;
      sum -== poisson * (1 - pow(q / p, z - k));
   }
   return sum;
}
```

با اجرای برخی از نتایج می‌توانیم افت احتمال نمایی با z را مشاهده کنیم.

```
q=0.1
z=0 P=1.0000000
z=1 P=0.2045873
z=2 P=0.0509779
z=3 P=0.0131722
z=4 P=0.0034552
z=5 P=0.0009137
z=6 P=0.0002428
z=7 P=0.0000647
z=8 P=0.0000173
z=9 P=0.0000046
```

```
z=10 P=0.0000012

q=0.3
z=0 P=1.0000000
z=5 P=0.1773523
z=10 P=0.0416605
z=15 P=0.0101008
z=20 P=0.0024804
z=25 P=0.0006132
```

با حل برحسب p کمتر از 1% داریم:

```
P < 0.001
q=0.10 z=5
q=0.15 z=8
q=0.20 z=11
q=0.25 z=15
q=0.30 z=24
q=0.35 z=41
q=0.40 z=89
q=0.45 z=340
```

نتیجه‌گیری

ما سیستمی برای تراکنش‌های الکترونیکی بدون تکیه به اعتماد پیشنهاد داده‌ایم. برای این منظور از چارچوب معمول سکه‌های ساخته شده از امضاهای دیجیتالی آغاز کردیم که کنترل قدرتمندی را در اختیار شما قرار می‌دهد، اما بدون راهی برای جلوگیری از خرج دوباره ناقص است. برای حل این مشکل، یک شبکه‌ی نظیر به نظیر، با استفاده از اثبات کار، را برای ثبت تاریخچه‌ای عمومی از تراکنش‌ها پیشنهاد دادیم؛ تراکنش‌هایی که اگر گره‌های درستکار بیشتر توان پردازنده را در اختیار خود داشته باشند، به لحاظ محاسباتی، تغییر دادن آن‌ها توسط مهاجم به سرعت محال می‌شود. قدرت این شبکه در سادگی ساخت‌نیافتگی آن نهفته است. گره‌ها همه به یک‌باره و با هماهنگی اندک کار می‌کنند. نیازی به شناسایی آن‌ها نیست، زیرا پیام‌ها به مکان خاصی منتقل نمی‌شوند و فقط باید بر اساس بهترین تلاش تحویل داده شوند. گره‌ها می‌توانند به خواست خود شبکه را ترک کنند، دوباره به کار خود بپردازند و به عنوان اثبات آن‌چه در غیابشان رخ داده است، زنجیر اثبات کار را بپذیرند. این گره‌ها با توان پردازنده‌ای خود رأی می‌دهند و پذیرش بلاک‌های معتبر را با تلاش برای بسط دادن آن‌ها و رد بلاک‌های نامعتبر را با امتناع از کار بر روی آن‌ها ابراز می‌کنند. با این راهکار اجماع، هرگونه قواعد و مشوق‌های لازم قابل اعمال است.

منابع

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In 20th Symposium on Information Theory in the Benelux, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In Journal of Cryptology, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In Sequences II: Methods in Communication, Security and Computer Science, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In Proceedings of the 4th ACM Conference on Computer and Communications Security, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure" http://www.hash-cash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications" 1957.

مجوز

این گزارش معتبر در اکتبر ۲۰۰۸ توسط ساتوشی ناکاموتو منتشر شد. بعداً (۲۰۰۹) به عنوان بخشی از مستندات پشتیبان به نرم‌افزار بیت‌کوین اضافه شد و دارای همان مجوز MIT است. در این کتاب، بدون هرگونه اصلاح، تحت شرایط مجوز MIT از آن نسخه‌برداری شده است:

پیوست ۲
عملگرها، ثابت‌ها و نمادهای اسکریپت تراکنش

جدول‌ها و توصیفات از منبع زیر برگرفته شده‌اند:
https://en.bitcoin.it/wiki/Script

جدول ب-۱ عملگرهای مربوط به نشاندن مقادیر روی پشته‌ها را نشان می‌دهد.

جدول ب-۱ نشاندن مقادیر روی پشته‌ها.

نماد	مقدار (هگزادسیمال)	شرح
OP_0 or OP_FALSE	0x00	آرایه‌ای خالی روی پشته قرار بده.
1-75	0x01-0x4b	N بایت بعدی را روی پشته قرار بده که N می‌تواند از ۱ تا ۷۵ بایت باشد.
OP_PUSHDATA1	0x4c	بایت اسکریپت بعدی حاوی N است، N بایت بعدی را روی پشته قرار بده.
OP_PUSHDATA2	0x4d	دو بایت اسکریپت بعدی حاوی N هستند، N بایت بعدی را روی پشته قرار بده.
OP_PUSHDATA4	0x4e	چهار بایت اسکریپت بعدی حاوی N هستند، N بایت بعدی را روی پشته قرار بده.
OP_1NEGATE	0x4f	مقدار 1- را روی پشته می‌نشاند.
OP_RESERVED	0x50	توقف کن - تراکنشِ نامعتبر، مگر این‌که در بخش اجرانشده‌ای از OP_IF یافت شود.
OP_1 or OP_TRUE	0x51	مقدار 1 را روی پشته قرار بده.
OP_2 to OP_16	0x52 to 0x60	برای OP_N، مقدار "N" را روی پشته قرار بده: به عنوان مثال، OP_2 مقدار "2" را در پشته قرار می‌دهد.

جدول ب-۲ عملگرهای کنترل جریان‌های شرطی را نشان می‌دهد.

جدول ب-۲ کنترل جریان‌های شرطی.

نماد	مقدار (هگزادسیمال)	شرح
OP_NOP	0x61	کاری نکن.
OP_VER	0x62	توقف کن - تراکنشِ نامعتبر، مگر اینکه در بخش اجرانشده‌ای از OP_IF یافت شود.
OP_IF	0x63	اگر بالای پشته 0 نیست، عبارت‌های بعدی را اجرا کن.
OP_NOTIF	0x64	اگر بالای پشته 0 است، عبارت‌های بعدی را اجرا کن.
OP_VERIF	0x65	توقف - تراکنش نامعتبراست.
OP_VERNOTIF	0x66	توقف - تراکنش نامعتبراست.
OP_ELSE	0x67	فقط در صورت عدم اجرای عبارت‌های بعدی قبلی، اجرا کن.
OP_ENDIF	0x68	بلاک OP_IF، OP_NOTIF، OP_ELSE را پایان بده.
OP_VERIFY	0x69	اگر TRUE نیست، بالای پشته را چک کن، توقف و تراکنش را بی اعتبار کن.
OP_RETURN	0x6a	توقف و تراکنش را بی اعتبار کن.

جدول ب-۳ عملگرهای به کار رفته برای قفل زمانی را نشان می‌دهد.

جدول ب-۳ عمل‌های قفل زمانی.

نماد	مقدار (هگزادسیمال)	شرح
OP_CHECKLOCKTIMEVERIFY (قبلاً OP_NOP2 بود)	0xb1	اگر آیتم بالایی پشته بزرگ‌تر از فیلد nLockTime تراکنش باشد، تراکنش را به عنوان تراکنشی نامعتبر علامت می‌زند، در غیر این صورت ارزیابی اسکریپت طوری ادامه می‌یابد که گویی OP_NOP اجرا شده است. افزون بر این، تراکنش نامعتبر قلمداد می‌شود اگر (۱) پشته خالی باشد؛ (۲) آیتم بالایی پشته منفی باشد؛ یا (۳) آیتم بالایی پشته بزرگ‌تر یا مساوی 500000000 باشد در حالی که فیلد nLockTime تراکنش کمتر از 500000000 باشد یا بالعکس؛ یا (۴) فیلد nSequency ورودی برابر 0xffff باشد. معانی دقیق در BIP-65 شرح داده شده است.
OP_CHECKSEQUENCEVERIFY (قبلاً OP_NOP3 بود)	0xb2	اگر زمان قفل نسبی ورودی (که توسط BIP 0068 با nSequence اعمال می‌شود) برابر یا بیشتر از مقدار آیتم بالایی پشته نباشد، تراکنش را به عنوان تراکنشی نامعتبر علامت می‌زند. معانی دقیق در BIP-112 شرح داده شده است.

جدول ب-۴ عملگرهای به کار رفته در دستکاری پشته را نشان می‌دهد.

جدول ب-۴ عمل‌های پشته.

نماد	مقدار (هگزادسیمال)	شرح
OP_TOALTST	0x6b	آیتم بالایی این پشته را بردار و بر پشته‌ای دیگر قرار بده.
OP_FROMAL	0x6c	آیتم بالایی پشته‌ی دیگری را بردار و در این پشته قرار بده.
OP_2DROP	0x6d	دو آیتم بالایی پشته را حذف کن.
OP_2DUP	0x6e	دو آیتم بالایی پشته را دوبرابر کن.
OP_3DUP	0x6f	سه آیتم بالایی پشته را دوبرابر کن.
OP_2OVER	0x70	آیتم‌های سوم و چهارم پشته را در بالای پشته کپی کن.
OP_2ROT	0x71	موارد پنجم و ششم پشته را به بالای پشته منتقل کن.
OP_2SWAP	0x72	دو جفت آیتم بالایی پشته را با هم تبادل کن.
OP_IFDUP	0x73	آیتم بالایی پشته را اگر 0 نیست، دوبرابر کن.
OP_DEPTH	0x74	آیتم‌های پشته را بشمار و تعداد حاصل را در پشته قرار بده.
OP_DROP	0x75	آیتم بالایی پشته را حذف کن.
OP_DUP	0x76	آیتم بالایی پشته را دوبرابر کن.
OP_NIP	0x77	آیتم دوم پشته را بردار.
OP_OVER	0x78	آیتم دوم را در پشته کپی کن و آن را بالای پشته قرار بده.
OP_PICK	0x79	مقدار N را از بالا بردار و سپس آیتم Nام را در بالای پشته کپی کن.
OP_ROLL	0x7a	مقدار N را از بالا بردار و سپس آیتم Nام را به بالای پشته منتقل کن.
OP_ROT	0x7b	سه آیتم بالایی پشته را بچرخان.
OP_SWAP	0x7c	سه آیتم بالایی پشته را با هم جابه‌جا کن.
OP_TUCK	0x7d	آیتم بالایی را کپی و آن را میان آیتم بالا و دومی درج کن.

جدول ب-۵ عملگرهای رشته‌ای را نشان می‌دهد.

جدول ب-۵ عمل‌های پیوندی رشته‌ها.

نماد	مقدار (هگزادسیمال)	شرح
OP_CAT	0x7e	غیرفعال (دو آیتم بالایی را به هم پیوند می‌زند).
OP_SUBST	0x7f	غیرفعال (زیررشته را بر می‌گرداند).
OP_LEFT	0x80	غیرفعال (زیررشته‌ی چپی را بر می‌گرداند).
OP_RIGHT	0x81	غیرفعال(زیررشته‌ی راستی را بر می‌گرداند).
OP_SIZE	0x82	طول رشته آیتم بالایی را محاسبه کن و نتیجه را در پشته قرار بده.

جدول ب-۶ عملگرهای جبر دودویی و منطق بولی را نشان می‌دهد.

جدول ب-۶ عمل‌های جبر دودویی و شرطی.

نماد	مقدار (هگزادسیمال)	شرح
OP_INVERT	0x83	غیرفعال (بیت‌های آیتم بالایی را وارونه کن)
OP_AND	0x84	غیرفعال (AND بولی دو آیتم بالایی)
OP_OR	0x85	غیرفعال (OR بولی دو آیتم بالایی)
OP_XOR	0x86	غیرفعال (XOR بولی دو آیتم بالایی)
OP_EQUAL	0x87	اگر دو آیتم بالایی دقیقاً برابرند، (1) TRUE را بنشان. در غیر این صورت، (0) FALSE را بنشان.
OP_EQUALVERIFY	0x88	همانند OP_EQUAL، اما اگر TRUE نبود، OP_VERIFY را تا توقف اجرا کن.
OP_RESERVED1	0x89	توقف کن - تراکنشِ نامعتبر، مگر این‌که در بخش اجرانشده‌ای از OP_IF یافت شود.
OP_RESERVED2	0x8a	توقف کن - تراکنشِ نامعتبر، مگر اینکه در بخش اجرانشده‌ای از OP_IF یافت شود.

جدول ب-۷ عملگرهای عددی (حسابی) را نشان می‌دهد.

جدول ب-۷ عملگرهای عددی.

نماد	مقدار (هگزادسیمال)	شرح
OP_1ADD	0x8b	1 را به آیتم بالایی اضافه کن.
OP_1SUB	0x8c	1 را از آیتم بالایی تفریق کن.
OP_2MUL	0x8d	غیرفعال (آیتم بالایی را در 2 ضرب کن).
OP_2DIV	0x8e	غیرفعال (آیتم بالایی را بر 2 تقسیم کن).
OP_NEGATE	0x8f	علامت آیتم بالایی را وارونه کن.
OP_ABS	0x90	علامت آیتم بالایی را به مثبت تغییر بده.
OP_NOT	0x91	اگر آیتم بالایی 0 یا 1 بود، آن را وارونه کن، در غیر این صورت 0 را برگردان.
OP_0NOTEQUAL	0x92	اگر آیتم بالایی 0 یا 1 بود، آن را وارونه کن، در غیر این صورت 1 را برگردان.
OP_ADD	0x93	دو آیتم بالایی را بردار، آن‌ها را جمع کن و نتیجه را در پشته قرار بده.
OP_SUB	0x94	دو آیتم بالایی را بردار، اولی را از دومی تفریق کن و نتیجه را در پشته قرار بده.
OP_MUL	0x95	غیرفعال (دو آیتم بالایی را در هم ضرب کن).
OP_DIV	0x96	غیرفعال (آیتم دوم را بر آیتم اول تقسیم کن).
OP_MOD	0x97	غیرفعال (باقیمانده‌ی تقسیم آیتم دوم بر آیتم اول را تعیین کن).

OP_LSHIFT	0x98	غیرفعال (آیتم دوم را به اندازه‌ی تعداد بیت‌های آیتم اول به سمت چپ جابه‌جا کن).
OP_RSHIFT	0x99	غیرفعال (آیتم دوم را به اندازه‌ی تعداد بیت‌های آیتم اول به سمت راست جابه‌جا کن).
OP_BOOLAND	0x9a	AND بولی دو آیتم بالایی.
OP_BOOLOR	0x9b	OR بولی دو آیتم بالایی.
OP_NUMEQUAL	0x9c	اگر دو آیتم بالایی، اعدادی برابرند، TRUE را برگردان.
OP_NUMEQUAL VERIFY	0x9d	همانند NUMEQUAL، سپس اگر TRUE نیست، OP_VERIFY را تا توقف اجرا کن.
OP_NUMNOTEQUAL	0x9e	اگر دو آیتم بالایی اعدادی برابر نیستند، TRUE را برگردان.
OP_LESSTHAN	0x9f	اگر آیتم دوم کوچکتر از آیتم بالایی است، TRUE را برگردان.
OP_GREATERTHAN	0xa0	اگر آیتم دوم بزرگتر ازآیتم بالایی است، TRUE را برگردان.
OP_LESSTHANOREQUAL	0xa1	اگر آیتم دوم کوچکتر یا مساوی با آیتم بالایی است، TRUE را برگردان.
OP_GREATERTHANOREQUAL	0xa2	اگر آیتم دوم بزرگتر یا مساوی با آیتم بالایی است، TRUE را برگردان.
OP_MIN	0xa3	از دو آیتم بالایی مورد کوچکتر را برگردان.
OP_MAX	0xa4	از دو آیتم بالایی مورد بزرگتر را برگردان.
OP_WITHIN	0xa5	اگر مورد سوم بین آیتم دوم (یا برابر با آن) و آیتم اول است، TRUE را برگردان.

جدول ب-۸ عملگرهای رمزنگاری را نشان می‌دهد.

جدول ب-۸ عملگرهای رمزنگاری و درهم‌سازی.

نماد	مقدار (هگزادسیمال)	شرح
OP_RIPEMD160	0xa6	درهم‌سازی RIPEMD160 را برای آیتم بالایی برگردان.
OP_SHA1	0xa7	درهم‌سازی SHA1 را برای آیتم بالایی برگردان.
OP_SHA256	0xa8	درهم‌سازی SHA256 را برای آیتم بالایی برگردان.
OP_HASH160	0xa9	درهم‌سازی RIPEMD160(SHA256(x)) را برای آیتم بالایی برگردان.
OP_HASH256	0xaa	درهم‌سازی SHA256(SHA256(x)) را برای آیتم بالایی برگردان.
OP_CODESEPARATOR	0xab	شروع داده‌های چک شده با امضا را علامت بزن.

OP_CHECKSIG	0xac	یک کلید عمومی و امضا را بردار و امضا را برای داده‌های درهم‌سازی‌شده‌ی تراکنش اعتبارسنجی کن. در صورت همخوانی، TRUE را برگردان.
OP_CHECKSIGVERIFY	0xad	همانند CHECKSIG، سپس اگر TRUE نبود، OP_VERIFY را تا توقف اجرا کن.
OP_CHECKMULTISIG	0xae	برای هر جفت امضا و کلید عمومی ارائه شده، CHECKSIG را اجرا کن. همه باید همخوانی داشته باشند.
OP_CHECKMULTISIGVERIFY	0xaf	همانند CHECKMULTISIG، سپس اگر TRUE نبود، OP_VERIFY را تا توقف اجرا کن.

جدول ب-۹ نمادهای غیر عملگری را نشان می‌دهد.

جدول ب-۹ نمادهای غیر عملگری.

نماد	مقدار (هگزادسیمال)	شرح
OP_NOP1-OP_NOP10	0xb0-0xb9	کاری انجام نمی‌دهد. از آن چشم‌پوشی می‌شود.

جدول ب-۱۰ کدهای عملگری رزروشده برای استفاده در اسکریپت تجزیه‌گر داخلی را نشان می‌دهد.

جدول ب-۱۰ کدهای OP رزروشده برای استفاده در اسکریپت تجزیه‌گر داخلی.

نماد	مقدار (هگزادسیمال)	شرح
OP_SMALLDATA	0xf9	فیلد داده‌های کوچک را نشان می‌دهد.
OP_SMALLINTEGER	0xfa	فیلد داده‌های عدد صحیح کوچک را نشان می‌دهد.
OP_PUBKEYS	0xfb	فیلدهای کلید عمومی را نشان می‌دهد.
OP_PUBKEYHASH	0xfd	یک فیلد درهم‌سازی کلید عمومی را نشان می‌دهد.
OP_PUBKEY	0xfe	یک فیلد کلید عمومی را نشان می‌دهد.
OP_INVALIDOPCODE	0xff	هر کد OP را که در حال حاضر اختصاص داده نشده است، نشان می‌دهد.

پیوست ۳ پیشنهادهای بهبود بیت‌کوین

پیشنهادهای بهبود بیت‌کوین اسناد طراحی‌ای هستند که اطلاعاتی را برای جامعه‌ی بیت‌کوین یا برای توصیف ویژگی جدید بیت‌کوین یا فرآیندها یا محیط آن فراهم می‌سازند. طبق اهداف و دستورالعمل‌های BIP که در BIP-01 ذکر شده است، سه نوع BIP وجود دارد:

BIP استاندارد

هر تغییری را توصیف می‌کند که بیشتر یا همه‌ی پیاده‌سازی‌های بیت‌کوین، مانند تغییر پروتکل شبکه یا تغییر در قوانین اعتبار تراکنش‌ها را تحت تأثیر قرار می‌دهد یا هرگونه تغییر یا اضافاتی را توصیف می‌کند که بر قابلیت همکاری متقابل اپلیکیشن‌ها با استفاده از بیت‌کوین تأثیر می‌گذارد.

BIP اطلاعاتی

یک مسأله‌ی طراحی بیت‌کوین را توصیف می‌کند، یا دستورالعمل‌ها یا اطلاعات کلی را در اختیار جامعه بیت‌کوین قرار می‌دهد، اما ویژگی جدیدی را پیشنهاد نمی‌کند. BIPهای اطلاعاتی لزوماً توصیه یا اجماع یک جامعه‌ی بیت‌کوین را نشان نمی‌دهند، بنابراین کاربران و مجریان ممکن است BIPهای اطلاعاتی را نادیده بگیرند یا توصیه‌های آن‌ها را دنبال کنند.

BIP فرآیند

یک فرآیند بیت‌کوین را توصیف می‌کند، یا تغییر (یا رویدادی در) یک فرآیند را پیشنهاد می‌کند. BIPهای فرآیند مانند BIPهای استاندارد هستند اما در حیطه‌هایی غیر از خود پروتکل بیت‌کوین کاربرد دارند. این BIPها ممکن است یک پیاده‌سازی را پیشنهاد کنند، اما نه به پایگاه کدهای بیت‌کوین. چنین BIPهایی غالباً مستلزم اجماع جامعه‌اند؛ و برخلاف BIPهای اطلاعاتی، چیزی بیشتر از توصیه هستند و کاربران معمولاً مجاز به نادیده گرفتن آن‌ها نیستند. مثال‌ها عبارتند از رویه‌ها، دستورالعمل‌ها، تغییراتی در فرآیند تصمیم‌گیری و تغییر در ابزارها یا محیط به کار رفته در توسعه‌ی بیت‌کوین. هر شبه BIP نیز یک BIP فرآیند محسوب می‌شود.

BIPها در مخزنی با نسخه‌ی معین در GitHub ضبط می‌شوند: https://github.com/bitcoin/bips.
در جدول پ-۱، تصاویری از BIPها در آوریل ۲۰۱۷ نشان داده شده است. برای اطلاعات به‌روز در مورد BIPهای موجود و محتوای آن‌ها به مخزن معتبر رجوع کنید.

جدول پ-۱ تصویری از BIPها.

شماره BIP	عنوان	مالک	نوع	وضعیت
BIP-1	اهداف و دستورالعمل‌های BIP	امیر تاکی[1]	فرآیند	جایگزین‌شده
BIP-2	فرآیند BIP، بازبینی‌شده	لوک دشیر[2]	فرآیند	فعال
BIP-8	بیت‌های نسخه با قفل درونی	شائولین فرای[3]	اطلاعاتی	پیش‌نویس
BIP-9	بیت‌های نسخه با زمان اضافه و تأخیر	پیتر وویل[4]، پیتر تاد[5]، گرگ مکسول[6]، راستی راسل[7]	اطلاعاتی	نهایی
BIP-10	توزیع تراکنش‌های چندامضایی	آلن راینر[8]	اطلاعاتی	پس‌گرفته‌شده
BIP-11	تراکنش‌های استاندارد M از N	گوین اندرسون[9]	استاندارد	نهایی
BIP-12	OP_EVAL	گوین اندرسون	استاندارد	پس‌گرفته‌شده
BIP-13	فرمت آدرس برای درهم‌سازی پرداخت به اسکریپت	گوین اندرسون	استاندارد	نهایی
BIP-14	نسخه‌ی پروتکل و عامل کاربری	امیر تاکی، پاتریک استریتمن[10]	استاندارد	نهایی
BIP-15	نام‌های مستعار	امیر تاکی	استاندارد	معوق
BIP-16	درهم‌سازی پرداخت به اسکریپت	گوین اندرسون	استاندارد	نهایی
BIP-17	OP_CHECKHASHVE (RIFY (CHV)	لوک دشیر	استاندارد	پس‌گرفته‌شده
BIP-18	hashScriptCheck	لوک دشیر	استاندارد	پیشنهادشده
BIP-19	تراکنش‌های استاندارد M از N (SigOp پایین)	لوک دشیر	استاندارد	پیش‌نویس
BIP-20	الگوی URI	لوک دشیر	استاندارد	جایگزین‌شده
BIP-21	الگوی URI	نیلز اشنایدر[11]، مت کورالو[12]	استاندارد	نهایی
BIP-22	getblocktemplate - مبانی	لوک دشیر	استاندارد	نهایی
BIP-23	getblocktemplate – استخراج مخزنی	لوک دشیر	استاندارد	نهایی
BIP-30	تراکنش‌های مضاعف	پیتر وویل	استاندارد	نهایی
BIP-31	پیام Pong	مایک هیرن[13]	استاندارد	نهایی
BIP-32	کیف پول قطعی سلسله مراتبی	پیتر وویل	استاندارد	نهایی
BIP-33	گره‌های استارت زده شده	امیر تاکی	استاندارد	پیش‌نویس
BIP-34	بلاک v2، ارتفاع در پایگاه سکه	گوین اندرسون	استاندارد	نهایی

1. Amir Taaki 2. Luke Dashjr 3. Shaolin Fry 4. Pieter Wuille 5. Peter Todd
6. Greg Maxwell 7. Rusty Russell 8. Alan Reiner 9. Gavin Andresen 10. Patrick Strateman
11. Nils Schneider 12. Matt Corallo 13. Mike Hearn

نهایی	استاندارد	جف گارزیک[1]	پیام مخزن حافظه	BIP-35
پیش‌نویس	استاندارد	استفن تامس[2]	سرویس‌های سفارشی	BIP-36
نهایی	استاندارد	مایک هیرن، مت کورالو	فیلترینگ اتصال بلوم	BIP-37
پیش‌نویس	استاندارد	مایک کالدول[3]، ارون وویزین[4]	کلید خصوصی محافظت شده با عبارت عبور	BIP-38
پیشنهادشده	استاندارد	مارِک پالاتینوس[5]، پاوول روزناک[6]، ارون وویزین، شان بو[7]	کد یادآوری برای تولید کلیدهای قطعی	BIP-39
شماره BIP اختصاص داده شده	استاندارد	مارِک پالاتینوس	پروتکل سیم لایه‌ای	BIP-40
شماره BIP اختصاص داده شده	استاندارد	مارِک پالاتینوس	پروتکل استخراج لایه‌ای	BIP-41
پیش‌نویس	استاندارد	پیتر وویل	یک منبع پولی متناهی برای بیت‌کوین	BIP-42
پیش‌نویس	اطلاعاتی	مارِک پالاتینوس، پاوول روزناک	فیلد پیشنهادی برای کیف پول‌های قطعی	BIP-43
پیشنهادشده	استاندارد	مارِک پالاتینوس، پاوول روزناک	سلسله مراتب حساب‌های چندگانه برای کیف پول‌های قطعی	BIP-44
پیشنهادشده	استاندارد	مانوئل آرائوز[8]، رایان اکس. چارلز[9]، ماتیاس آلخو گارسیا[10]	ساختار مربوط به کیف پول‌های قطعی چندامضایی P2SH	BIP-45
پیش‌نویس	اطلاعاتی	یوستوس رانفیر[11]	کدهای پرداخت با قابلیت استفاده‌ی مجدد برای کیف پول‌های قطعی سلسله مراتبی	BIP-47
پیش‌نویس	اطلاعاتی	دانیل ویگل[12]	الگوی به‌دست آوردن حساب‌های مبتنی بر P2WPKH لانه‌ای در P2SH	BIP-49
نهایی	اطلاعاتی	گَوین اندرسون	چین فورک پس از مرگ (مارس ۲۰۱۳)	BIP-50
پیش‌نویس	استاندارد	امیر تاکی	پیام «نسخه» با طول ثابت (فیلد تراکنش‌های بازپخشی)	BIP-60
نهایی	استاندارد	گَوین اندرسون	پیام رد نظیر به نظیر	BIP-61
پس‌گرفته‌شده	استاندارد	پیتر وویل	مقابله با تغییرپذیری	BIP-62
شماره BIP اختصاص داده شده	استاندارد	پیتر تاد	آدرس‌های پنهانی	BIP-63
پیش‌نویس	استاندارد	مایک هیرن	پیام getutxo	BIP-64
نهایی	استاندارد	پیتر تاد	OP_CHECKLOCKTIMEVERIFY	BIP-65

1. Jef Garzik 2. Stefan Thomas 3. Mike Caldwell 4. Aaron Voisine
5. Marek Palatinus 6. Pavol Rusnak 7. Sean Bowe 8. Manuel Araoz
9. Ryan X. Charles 10. Matias Alejo Garcia 11. Justus Ranvier 12. Daniel Weigl

BIP-66	امضاهای DER اکید	پیتر وویل	استاندارد	نهایی
BIP-67	آدرس‌های چندامضایی پرداخت قطعی به درهم‌سازی اسکریپت از طریق مرتب‌سازی کلید عمومی	تامس کرین[1]، ژان-پیر روپ[2]، روبن دو وری[3]	استاندارد	پیشنهادشده
BIP-68	زمان قفل نسبی با استفاده از اعداد ترتیبی ناشی از اجماع	مارک فریدنباخ[4]، بی‌تی‌سی دارک[5]، نیکولاس دوریه[6]، کینوشیتاجونا[7]	استاندارد	نهایی
BIP-69	نمایه‌سازی واژه‌نگاری ورودی‌ها و خروجی‌های تراکنش‌ها	کریستوف اتلس[8]	اطلاعاتی	پیشنهادشده
BIP-70	پروتکل پرداخت	گَوین اندرسون، مایک هیرن	استاندارد	نهایی
BIP-71	انواع MIME پروتکل پرداخت	گَوین اندرسون	استاندارد	نهایی
BIP-72	بیت‌کوین: الحاقیه‌های URI برای پروتکل پرداخت	گَوین اندرسون	استاندارد	نهایی
BIP-73	استفاده از هدر «پذیرش» برای چانه‌زنی نوع پاسخ با URLهای درخواست پرداخت	استفن پِیر[9]	استاندارد	نهایی
BIP-74	مجازسازی مقدار صفر OP_RETURN در پروتکل پرداخت	توبی پادیلا[10]	استاندارد	پیش‌نویس
BIP-75	تبادل آدرس‌های خارج از گروه با استفاده از رمزگردانی پروتکل	جاستین نیوتن[11]، مت دیوید[12]، ارون وویزین، جیمز مک‌وایت[13]	استاندارد	پیش‌نویس
BIP-80	سلسله‌مراتبی برای کیف پول‌های چندامضایی قطعی مخزن (رأی‌دهی غیررنگی)	یوستوس رانفیر، جیمی سانگ[14]	اطلاعاتی	معوق
BIP-81	سلسله‌مراتبی برای کیف پول‌های چندامضایی قطعی مخزن (رأی‌دهی رنگی)	یوستوس رانفیر، جیمی سانگ	اطلاعاتی	معوق
BIP-83	درخت کلیدهای قطعی سلسله‌مراتبی دینامیک	اریک لومبروزو[15]	استاندارد	پیش‌نویس
BIP-90	استقرارهای دفن‌شده	سوهاس دفتوار[16]	اطلاعاتی	پیش‌نویس
BIP-99	انگیزش و استقرار تغییرات به‌عمل‌آمده در قواعد اجماع (چنگک‌های [نرم/سخت])	خورخه تیمون[17]	اطلاعاتی	پیش‌نویس
BIP-101	افزایش اندازه‌ی بیشینه‌ی بلاک	گوین اندرسون	استاندارد	پس‌گرفته‌شده
BIP-102	افزایش اندازه‌ی بلاک به دو مگابایت	جف گارزیک	استاندارد	پیش‌نویس

1. Thomas Kerin 2. Jean-Pierre Rupp 3. Ruben de Vries 4. Mark Friedenbach
5. BtcDrak 6. Nicolas Dorier 7. Kinoshitajona 8. Kristov Atlas 9. Stephen Pair
10. Toby Padilla 11. Justin Newton 12. Matt David 13. James MacWhyte
14. Jimmy Song 15. Eric Lombrozo 16. Suhas Daftuar 17. Jorge Timón

پیش‌نویس	استاندارد	پیتر وویل	اندازه‌ی بلاک در پی رشد فناوری	BIP-103
پیش‌نویس	استاندارد	ت.خان[1]	بلاک ۷۵- اندازه‌ی بیشینه‌ی بلاک	BIP-104
پیش‌نویس	استاندارد	بی‌تی‌سی دارک	الگوریتم هدف‌گیری دوباره‌ی مبتنی بر اجماع اندازه‌ی بلاک	BIP-105
پیش‌نویس	استاندارد	اوپال چاکرابورتی[2]	اندازه‌ی بیشینه‌ی بلاک بیت‌کوین با کنترل دینامیک	BIP-106
پیش‌نویس	استاندارد	واشینگتون وای سانچز[3]	حد دینامیک بر اندازه‌ی بلاک	BIP-107
مردود	استاندارد	گوین اندرسون	اندازه‌ی دو میلیون بایتی با حدهای sigop و sighash	BIP-109
پیشنهادشده	استاندارد	مت کورالو، پیتر تاد	بیت سرویس NODE_BLOOM	BIP-111
نهایی	استاندارد	بی‌تی‌سی دارک، مارک فریدنباخ، اریک لومبروزو	CHECKSEQUENCEVERIFY	BIP-112
نهایی	استاندارد	تامس کرین، مارک فریدنباخ	میانه‌ی زمان سپری شده به‌عنوان نقطه‌ی پایانی برای محاسبات قفل زمانی	BIP-113
پیش‌نویس	استاندارد	جانسون لائو[4]	درخت نحو انتزاعی مرکلی‌شده	BIP-114
پیش‌نویس	استاندارد	کاله روزنباوم[5]	اثبات پرداخت	BIP-120
پیش‌نویس	استاندارد	کاله روزنباوم	طرح URI برای اثبات پرداخت	BIP-121
پیش‌نویس	استاندارد	مارکو پونتلو[6]	طرح URI برای مراجع بلاک چین یا کندوکاو در آن‌ها	BIP-122
فعال	فرآیند	اریک لومبروزو	طبقه‌بندی BIPها	BIP-123
پیش‌نویس	اطلاعاتی	اریک لومبروزو، ویلیام سوانسون[7]	الگوهای اسکریپت قطعی سلسله‌مراتبی	BIP-124
پیشنهادشده	استاندارد	پیتر تاد، دیوای هاردینگ[8]	سیگنال‌دهی برای جایگزینی با اجرت	BIP-125
پیش‌نویس	اطلاعاتی	کریستوف اتلس	بهترین اقدامات برای تراکنش‌های ورودی ناهمگن	BIP-126
پیشنهادشده	استاندارد	سوهاس دفتوار	پیام هدر ارسال	BIP-130
پیش‌نویس	استاندارد	کریس پریست[9]	مشخص‌سازی «تراکنش‌های ادغام‌گر» (ورودی‌های کاراکتر عام)	BIP-131
پس‌گرفته‌شده	فرآیند	اندی چِیس[10]	فرآیند پذیرش BIPهای کمیته-محور	BIP-132
پیش‌نویس	استاندارد	الکس مارکوس[11]	پیام فیلتر رایگان	BIP-133

1. t.khan 2. Upal Chakraborty 3. Washington Y. Sanchez 4. Johnson Lau
5. Kalle Rosenbaum 6. Marco Pontello 7. William Swanson 8. David A. Harding
9. Chris Priest 10. Andy Chase 11. Alex Morcos

BIP-134	تراکنش‌های انعطاف‌پذیر	تام زندر[1]	استاندارد	پیش‌نویس
BIP-140	TXID بهنجارشده	کریستین دِکِر[2]	استاندارد	پیش‌نویس
BIP-141	شاهد جداسازی‌شده (لایه‌ی اجماع)	اریک لومبروزو، جانسون لائو، پیتر وویل	استاندارد	پیش‌نویس
BIP-142	فرمت آدرس برای شاهد جداسازی‌شده	جانسون لائو	استاندارد	معوق
BIP-143	صحه‌گذاری بر امضای تراکنش برای برنامه‌ی شاهد نسخه‌ی 0	جانسون لائو، پیتر وویل	استاندارد	پیش‌نویس
BIP-144	شاهد جداسازی‌شده (سرویس‌های نظیر)	اریک لومبروزو، پیتر وویل	استاندارد	پیش‌نویس
BIP-145	بهنگام‌سازی‌های getblocktemplate برای شاهد جداسازی‌شده	لوک دشیر	استاندارد	پیش‌نویس
BIP-146	مقابله با تغییرپذیری رمزگردانی امضا	جانسون لائو، پیتر وویل	استاندارد	پیش‌نویس
BIP-147	مقابله با تغییرپذیری عنصر ساختگی در پشته	جانسون لائو	استاندارد	پیش‌نویس
BIP-148	فعال‌سازی اجباری استقرار segwit	شائولین فرای	استاندارد	پیش‌نویس
BIP-150	تأیید نظیر	یوناس اشنلی[3]	استاندارد	پیش‌نویس
BIP-151	رمزگردانی ارتباطات نظیر به نظیر	یوناس اشنلی	استاندارد	پیش‌نویس
BIP-152	بازپخش بلاک فشرده	مت کورالو	استاندارد	پیش‌نویس
BIP-171	رابط برنامه‌نویسی کاربردی اطلاعات تبادل ارزی	لوک دشیر	استاندارد	پیش‌نویس
BIP-180	مقابله با جعل اندازه/وزن بلاک	لوک دشیر	استاندارد	پیش‌نویس
BIP-199	تراکنش های قراردادی درهم‌سازی‌شده با قفل زمانی	شان بو، دیرا هاپوود[4]	استاندارد	پیش‌نویس

1. Tom Zander 2. Christian Decker 3. Jonas Schnelli 4. Daira Hopwood

پیوست ۴ شاهد تفکیک‌شده

شاهد تفکیک‌شده[1] (segwit) نسخه‌ی به‌هنگام‌شده‌ای از قوانین اجماع بیت‌کوین و پروتکل شبکه است که به عنوان یک انشعاب نرم BIP-9 که در حال حاضر (اواسط ۲۰۱۷) منتظر فعال‌سازی است، پیشنهاد و پیاده‌سازی می‌شود.

در رمزنگاری، اصطلاح "شاهد" برای توصیف راه‌حلی برای یک معمای رمزنگاری استفاده می‌شود. در حیطه‌ی بیت‌کوین، شاهد در یک شرط رمزنگاری که بر روی خروجی خرج‌نشده‌ی تراکنش (UTXO) اعمال می‌شود، صدق می‌کند.

در حیطه‌ی بیت‌کوین، امضای دیجیتالی یکی از انواع شاهد است، اما شاهد در مفهوم گسترده‌تر آن عبارت از هر حلی است که بتواند شرایط تحمیل شده بر یک UTXO را برآورده کند و آن UTXO را برای خرج کردن، باز کند. اصطلاح "شاهد" اصطلاح عمومی‌تری برای "اسکریپت قفل‌گشا" یا scriptSig است.

پیش از معرفی segwit، هر ورودی در یک تراکنش توسط داده‌های شاهدی که قفل آن را باز کرده بودند، دنبال می‌شد. داده‌های شاهد به عنوان بخشی از هر ورودی در تراکنش تعبیه می‌شد. اصطلاح شاهد تفکیک شده، یا به طور خلاصه segwit، بر تفکیک امضا یا گشودن قفل اسکریپت از یک خروجی خاص دلالت دارد. به "scriptSi جداگانه" یا "امضای جداگانه" در ساده‌ترین شکل فکر کنید. بنابراین شاهد تفکیک‌شده، تغییری در معماری بیت‌کوین است که هدف آن انتقال داده‌های شاهد از قسمت scriptSig (اسکریپت قفل‌گشای) یک تراکنش به یک ساختار داده‌ی شاهد تفکیک‌شده است که با تراکنشی همراه است. کلاینت‌ها می‌توانند داده‌های تراکنش را با یا بدون اطلاعات شاهد همراه درخواست کنند.

در این بخش به برخی از فواید شاهد تفکیک‌شده می‌پردازیم، راهکار به کار رفته برای استقرار و پیاده‌سازی این تغییر معماری را شرح می‌دهیم و استفاده از شاهد تفکیک‌شده را در تراکنش‌ها و آدرس‌ها نشان می‌دهیم.

1. Segregated Witness

شاهد تفکیک‌شده توسط BIPهای زیر تعریف می‌شود:

BIP-141

تعریف اصلی شاهد تفکیک‌شده.

BIP-143

صحت‌سنجی بر امضای تراکنش برای برنامه‌ی شاهد نسخه‌ی صفر

BIP-144

خدمات همتا- پیام‌های جدید شبکه و فرمت‌های سریال سازی

BIP-145

به‌هنگام‌سازی‌های getblocktemplate برای شاهد تفکیک‌شده (جهت استخراج)

چرا شاهد تفکیک‌شده؟

شاهد تفکیک‌شده تغییری در معماری است که تأثیرات زیادی بر مقیاس‌پذیری، امنیت، مشوق‌های اقتصادی و عملکرد بیت‌کوین دارد:

تغییرپذیری تراکنش

با انتقال شاهد به بیرون از تراکنش، درهم‌سازی تراکنشی که به عنوان شناسه استفاده می‌شود، دیگر شامل داده‌های شاهد نمی‌شود. از آن‌جا که داده‌های شاهد تنها بخشی از تراکنش است که توسط طرف سوم قابل تغییر است، حذف آن نیز فرصت حملات تغییرپذیری تراکنش را منتفی می‌کند. با شاهد تفکیک‌شده، تغییر دادن درهم‌سازی‌های تراکنشی توسط کسی غیر از ایجادکننده‌ی تراکنش، ناممکن می‌شود و این موضوع سبب بهبود کلی در پیاده‌سازی بسیاری از پروتکل‌های دیگری می‌شود که متکی بر ساخت‌وسازهای پیشرفته‌ی تراکنشی بیت‌کوین، مانند کانال‌های پرداخت، تراکنش‌های زنجیره‌ای و شبکه‌های آذرخش[1] هستند.

تعیین شماره‌ی نسخه‌ی اسکریپت

مشابه با تراکنش‌ها و بلاک‌ها که شماره‌ی نسخه دارند، با معرفی اسکریپت‌های شاهد تفکیک‌شده، پیش از هر اسکریپت قفل‌گذار، یک شماره نسخه‌ی اسکریپت می‌آید. افزودن شماره نسخه اسکریپت به زبان اسکریپت‌نویسی امکان ارتقا به شیوه‌ای سازگار با نسخه‌های پیشین را می‌دهد (یعنی با استفاده از ارتقاهای انشعاب نرم) تا بتواند عملگرهای اسکریپت، نحوی یا معانی جدید را معرفی کند. توانایی به‌روزرسانی زبان اسکریپت‌نویسی به روشی غیرقابل انکار باعث افزایش سرعت نوآوری در بیت‌کوین خواهد شد.

مقیاس‌بندی ذخیره‌سازی و شبکه

داده‌های شاهد غالباً سهم بزرگی در اندازه‌ی کلی یک تراکنش دارند. اسکریپت‌های پیچیده‌تر نظیر آن‌هایی که برای کانال‌های پرداخت یا چندامضایی استفاده می‌شوند، بسیار بزرگ‌اند. در برخی موارد،

1. Lightning networks

این اسکریپت‌ها اکثریت داده‌های یک تراکنش (بیش از ۷۵٪) را تشکیل می‌دهند. شاهد تفکیک‌شده با انتقال داده‌های شاهد به بیرون تراکنش، مقیاس‌پذیری بیت‌کوین را بهبود می‌بخشد. گره‌ها می‌توانند داده‌های شاهد را پس از اعتبارسنجی هرس کنند یا هنگام انجام تأیید پرداخت ساده‌شده، آن را یکسره نادیده بگیرند. نیازی به انتقال داده‌های شاهد به همه‌ی گره‌ها نیست و ذخیره‌سازی آن روی دیسک توسط همه‌ی گره‌ها ضرورتی ندارد.

بهینه‌سازی صحت‌سنجی امضا

شاهد تفکیک‌شده، توابع امضا (CHECKSIG، CHECKMULTISIG و غیره) را ارتقا می‌دهد تا از پیچیدگی محاسباتی الگوریتم بکاهد. پیش از segwit، الگوریتم به‌کاررفته برای تولید امضا به تعدادی عملیات درهم‌سازی نیاز داشت که متناسب با اندازه‌ی تراکنش بود. محاسبات درهم‌سازی داده‌ها با تعداد عملیات امضا به نسبت $O(n^2)$ افزایش می‌یافت و به تمام گره‌های تأیید امضا بار محاسباتی قابل توجهی را وارد می‌آورد. با segwit، الگوریتم به گونه‌ای تغییر می‌کند که پیچیدگی آن به $O(n)$ کاهش می‌یابد.

بهبودبخشیدن به امضای آفلاین

امضاهای شاهد تفکیک‌شده مقدار (مبلغ) ارجاع شده توسط هر ورودی را در درهم‌سازی امضا شده درج می‌کنند. قبلاً، یک دستگاه امضای آفلاین، مانند یک کیف پول سخت‌افزاری، باید مقدار هر ورودی را پیش از امضای تراکنش صحت‌سنجی می‌کرد. این کار معمولاً با به جریان درآوردن مقدار زیادی از داده‌های مربوط به تراکنش‌های قبلی انجام می‌شود که به عنوان ورودی ارجاع داده شده‌اند. از آن‌جا که مبلغ اکنون بخشی از درهم‌سازی تعهد امضا شده است، دستگاه آفلاین نیازی به تراکنش‌های قبلی ندارد. اگر مبالغ مطابقت نداشته باشند (توسط یک سیستم آنلاین لو رفته به نمایش در آیند)، امضا نامعتبر خواهد بود.

شیوه‌ی عملکرد شاهد تفکیک‌شده

در نگاه نخست، به نظر می‌رسد شاهد تفکیک‌شده تغییری در چگونگی ساخت تراکنش‌ها و در نتیجه یک ویژگی در سطح تراکنش باشد، اما چنین نیست. در حقیقت، شاهد تفکیک‌شده تغییر در چگونگی خرج شدن UTXO شخصی نیز هست و بنابراین یک ویژگی مرتبط خروجی به شمار می‌رود.

یک تراکنش می‌تواند خروجی‌های شاهد تفکیک‌شده یا خروجی‌های سنتی (شاهد درون‌خطی) یا هر دو را خرج کند. بنابراین، چندان منطقی نیست که از یک تراکنش با عنوان "تراکنش شاهد تفکیک‌شده" یاد شود. بلکه باید از ورودی‌های تراکنش‌های خاص با عنوان "ورودی‌های شاهد تفکیک‌شده" یاد کرد.

هنگامی که تراکنشی یک UTXO را خرج می‌کند، باید شاهدی ارائه دهد. در UTXO سنتی، اسکریپت قفل‌گذاری این موضوع را ضروری می‌کند که داده‌های شاهد در قسمت ورودی تراکنشی که

UTXO را خرج می‌کند، به صورت درون‌خطی[1] ارائه شود. اما UTXO شاهد تفکیک‌شده، اسکریپت قفل‌سازی را مشخص می‌کند که داده‌های شاهد خارج از ورودی (تفکیک شده) می‌تواند آن را برآورده کند.

انشعاب نرم (سازگاری با نسخه‌های پیشین)

شاهد تفکیک‌شده، تغییر قابل توجهی در شیوه‌ی معماری خروجی‌ها و تراکنش‌هاست. چنین تغییری معمولاً مستلزم تغییر همزمان در هر گره‌ی بیت‌کوین و تغییر به منظور تغییر در قوانین اجماع دارد – آن‌چه که به عنوان انشعاب سخت شناخته می‌شود.

در عوض، تغییرات ناشی از شاهد تفکیک‌شده اختلال بسیار کمتری را سبب می‌شود و با نسخه‌های پیشین سازگاری دارد که این حالت به عنوان یک انشعاب نرم شناخته می‌شود. در این نوع از ارتقا به نرم‌افزارهای ارتقانیافته می‌توان تغییرات را نادیده گرفت و این نرم افزارها بدون هیچ گونه اختلالی به کار خود ادامه دهند.

خروجی‌های شاهد تفکیک‌شده به گونه‌ای بنا می‌شوند که سیستم‌های قدیمی‌تری که از segwit آگاه نیستند، هنوز هم می‌توانند آن‌ها را اعتبارسنجی کنند. از نگاه یک کیف پول یا گره‌ی قدیمی، خروجی شاهد تفکیک‌شده شبیه به خروجی‌ای است که هرکسی می‌تواند آن را خرج کند. خروجی‌های جستجو را می‌توان با امضای خالی خرج کرد؛ پس این واقعیت که هیچ امضایی در تراکنش وجود ندارد (تفکیک شده است) تراکنش را بی اعتبار نمی‌کند. به هرحال، کیف پول‌های و گره‌های استخراج جدیدتر، خروجی شاهد تفکیک‌شده را می‌بینند و انتظار دارند در داده‌های شاهد تراکنش، شاهدی معتبری برای آن بیابند.

مثال‌هایی از خروجی شاهد تفکیک‌شده و تراکنش‌ها

اکنون چند مثال از تراکنش‌ها و چگونگی تغییر آن‌ها با شاهد تفکیک‌شده را بررسی می‌کنیم. نخست ببینیم پرداخت Pay-to-Public-Key-Hash (P2PKH) چگونه با برنامه‌ی شاهد تفکیک‌شده تبدیل می‌شود. سپس به هم‌ارز شاهد تفکیک‌شده برای اسکریپت‌های Pay-to-Script-Hash (P2SH) می‌پردازیم. سرانجام، خواهیم دید که چگونه برنامه‌های شاهد تفکیک‌شده را می‌توان درون یک اسکریپت P2SH ادغام کرد.

درهم‌سازی کلید عمومی پرداخت به شاهد (P2WPKH)

در مثال خرید یک فنجان قهوه، آلیس برای پرداخت وجه یک فنجان قهوه‌ی باب تراکنشی را ایجاد کرد. آن تراکنش یک خروجی P2PKH به ارزش 0.015 BTC ایجاد کرد که باب می‌توانست آن را خرج کند. اسکریپت خروجی به شرح زیر است:

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 EQUALVERIFY
CHECKSIG
```

1. Inline

آلیس در صورت استفاده از شاهد تفکیک‌شده، اسکریپ درهم‌سازی کلید عمومی پرداخت به شاهد (P2WPKH) زیر را ایجاد می‌کرد که ظاهری شبیه به این داشت:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

چنان‌که مشاهده می‌کنید، اسکریپت قفل‌گذاری خروجیِ شاهد تفکیک‌شده بسیار ساده‌تر از یک خروجی سنتی است. این اسکریپت شامل دو مقدار می‌شود که روی پشته‌ی ارزیابی اسکریپت قرار داده می‌شود. برای یک کلاینت قدیمی بیت‌کوین (ناآگاه از segwit) این دو مقدار موجود در پشته همانند یک خروجی به نظر می‌رسند که هر کسی می‌تواند آن را خرج کند و نیاز به امضا ندارد (یا به عبارتی می‌توان آن را با امضای خالی خرج کرد). از نگاه کلاینت جدید و آگاه از segwit نخستین شماره (0) به عنوان شماره نسخه (نسخه‌ی شاهد) تفسیر می‌شود و و بخش دوم (۲۰ بایت)، هم‌ارز اسکریپت قفل‌گذاری است و به عنوان برنامه‌ی شاهد شناخته می‌شود. این برنامه‌ی شاهد ۲۰ بایتی در واقع در هم‌سازی کلید عمومی در یک اسکریپت P2PKH است.

اکنون، به تراکنش نظیری که باب برای خرج کردن این خروجی استفاده می‌کند، نگاهی بیندازیم. برای اسکریپت اصلی (غیر segwit)، تراکنش باب باید امضایی را در ورودی تراکنش لحاظ کند:

```
[...]
"Vin" : [
"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
   "scriptSig": "<Bob's scriptSig>",
]
[...]
```

با این حال، برای خرج کردن خروجی شاهد تفکیک‌شده، تراکنش هیچ امضایی بر آن ورودی ندارد. در عوض، تراکنش باب دارای یک اسکریپت scriptSig است و شامل یک شاهد تفکیک‌شده‌ی خارج از خود تراکنش می‌شود:

```
"Vin" : [
"txid":
"0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
   "scriptSig": "",
]
[...]
"witness": "<Bob's witness data>"
[...]
```

ساخت کیف پول P2WPKH

این نکته اهمیت بسیار دارد که P2WPKH فقط باید توسط دریافت کننده (گیرنده) وجه ایجاد شود و توسط فرستنده از یک کلید عمومی شناخته شده، اسکریپت P2PKH یا آدرس تبدیل نشود. فرستنده هیچ راهی ندارد که بداند که آیا کیف پول گیرنده توانایی انجام تراکنش و خرج کردن خروجی‌های P2WPKH را دارد یا خیر.

علاوه‌براین، خروجی P2WPKH باید از درهم‌سازی کلیدهای عمومی فشرده‌شده بنا نهاده شود. کلیدهای عمومی فشرده نشده در segwit غیراستانداردند و ممکن است در آینده به صراحت توسط یک انشعاب نرم غیرفعال شوند. اگر منشأ درهم‌سازی در P2WPKH یک کلید عمومی فشرده نشده باشد، ممکن است قابل خرج کردن نباشد و ممکن است پول خود را از دست بدهید. خروجی‌های P2WPKH باید توسط کیف پول گیرنده و با به دست آوردن یک کلید عمومی از کلید خصوصی آن‌ها ایجاد شوند.

P2WPKH باید توسط دریافت کننده (گیرنده) وجه با با تبدیل کلید عمومی فشرده به درهم‌سازی P2WPKH ساخته شود. هرگز نباید یک اسکریپت P2PKH، آدرس بیت‌کوین یا کلید عمومی فشرده نشده را به یک اسکریپت شاهد P2WPKH تبدیل کنید.

درهم‌سازی اسکریپت پرداخت به شاهد (P2WSH)

نوع دوم برنامه‌ی شاهد، متناظر با درهم‌سازی اسکریپت پرداخت به اسکریپت (P2SH) است. این نوع اسکریپت را در بخش ۷-۳ دیدیم. در آن مثال، P2SH توسط شرکت محمد برای بیان یک اسکریپت چندامضایی استفاده شد. وجوه پرداخت شده به شرکت محمد با اسکریپت قفل‌گذاری شبیه به این رمزگذاری می‌شدند:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

این اسکریپت P2SH به درهم‌سازی یک اسکریپت وصول اشاره دارد که شرط چندامضایی دو امضا از سه امضا را برای خرج کردن پول تعریف می‌کند. برای خرج کردن این خروجی، شرکت محمد اسکریپت وصول (که درهم‌سازی آن با درهم‌سازی اسکریپت خروجی P2SH همخوانی دارد) و امضاهای لازم برای اسکریپت وصول را که همگی در داخل ورودی تراکنش وجود دارند، ارائه می‌دهد:

```
[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
   "scriptSig": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5
CHECKMULTISIG>",
]
```

اکنون، ببینیم چگونه می‌توان کل این مثال را به segwit ارتقا داد. اگر مشتریان محمد از یک کیف پول سازگار با segwit استفاده می‌کردند، می‌توانستند با ایجاد خروجی درهم‌سازی اسکریپت پرداخت به شاهد (H2WSH) به شکل زیر، پرداخت را انجام دهند:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

این بار نیز همانند مثال P2WPKH، مشاهده می‌کنید که اسکریپت هم‌ارز شاهد تفکیک‌شده بسیار ساده‌تر است و عملوندهای مختلف اسکریپت را که در اسکریپت‌های P2SH می‌بینید، حذف نمی‌کند. در عوض، برنامه‌ی شاهد تفکیک‌شده شامل دو مقدار موجود در بالای پشته می‌شود: یک نسخه‌ی شاهد و درهم‌سازی ۳۲ بایتی SHA256 اسکریپت وصول.

در حالی که P2SH از درهم‌سازی ۲۰ بایتی (RIPEMD160(SHA256 (script)) استفاده می‌کند، برنامه‌ی شاهد P2WSH از یک درهم‌سازی ۳۲ بایتی SHA256(script) استفاده می‌کند. این تفاوت در انتخاب الگوریتم درهم‌سازی عمدی است و برای تمایز میان دو نوع برنامه‌ی شاهد (P2WPKH و P2WSH) بر اساس طول درهم‌سازی و فراهم ساختن امنیت بیشتر برای P2WSH (۱۲۸ بیت در مقابل ۸۰ بیت P2SH) استفاده می‌شود.

شرکت محمد می‌تواند با ارائه‌ی اسکریپت وصول صحیح و امضاهای کافی برای برآورده کردن آن، خروجی P2WSH را خرج کند. اسکریپت بازخرید و امضاها، هر دو، در خارج از تراکنش خرج‌کردن به عنوان بخشی از داده‌های شاهد، تفکیک می‌شوند. کیف پول محمد در درون ورودی تراکنش، یک scriptSig خالی قرار می‌دهد:

```
[...]
"Vin" : [
"txid": "abcdef12345...",
"vout": 0,
   "scriptSig": "",
]
[...]
"witness": "<SigA> <SigB> <2 PubA PubB PubC PubD PubE 5
    CHECKMULTISIG>"
[...]
```

تمایز میان P2WPKH و P2WSH

در دو بخش قبل، دو نوع برنامه‌ی شاهد را نشان دادیم: درهم‌سازی کلید عمومی پرداخت به شاهد (P2WPKH) و درهم‌سازی اسکریپت پرداخت به شاهد (P2WSH). هر دو نوع برنامه‌ی شاهد شامل یک شماره نسخه بایت واحد می‌شوند که به دنبال آن یک درهم‌سازی طولانی‌تر می‌آید. این دو بسیار شبیه به هم به نظر می‌رسند، اما تفسیرهای بسیار متفاوتی از آن‌ها می‌شود: یکی به عنوان یک درهم‌سازی کلید عمومی تعبیر می‌شود که با امضا برآورده می‌شود و دیگری به عنوان درهم‌سازی اسکریپت که با یک اسکریپت وصول برآورده می‌شود. تفاوت مهم میان آن‌ها در طول درهم‌سازی است:

- طول درهم‌سازی کلید عمومی در P2WPKH، ۲۰ بایت است.
- طول درهم‌سازی اسکریپت در P2WSH، ۳۲ بایت است.

این تنها تفاوتی است که باعث می‌شود کیف پول میان دو نوع برنامه‌ی شاهد تفاوت قائل شود. کیف پول با نگاهی به طول درهم‌سازی می‌تواند تعیین کند چه نوع برنامه‌ای شاهد آن است، P2WPKH یا P2WSH.

ارتقا به شاهد تفکیک‌شده

چنان‌که از نمونه‌های قبلی پیداست، ارتقا به شاهد تفکیک‌شده یک فرآیند دو مرحله‌ای است. نخست، کیف پول‌ها باید خروجی‌هایی از نوع segwit ویژه ایجاد کنند. سپس، این خروجی‌ها را می‌توان با کیف پول‌هایی خرج کرد که می‌دانند چگونه تراکنش‌های شاهد تفکیک‌شده را بنا کنند. در مثال‌های

پیشین، کیف پول آلیس آگاه از segwit و قادر به تولید خروجی‌های ویژه با اسکریپت‌های شاهد تفکیک‌شده بود. کیف پول باب نیز آگاه از segwit و قادر به خرج کردن آن خروجی‌هاست. آن‌چه ممکن است از این مثال آشکار نباشد این است که در عمل، کیف پول آلیس باید بداند که باب از یک کیف پول آگاه از segwit استفاده می‌کند و توانایی خرج کردن این خروجی‌ها را دارد. در غیر این صورت، اگر کیف پول باب ارتقا داده نشود و آلیس سعی کند پرداخت‌هایش به باب را به شیوه‌ی segwit انجام دهد، کیف پول باب قادر به تشخیص این پرداخت‌ها نخواهد بود.

برای انواع پرداخت P2WPKH و P2WSH، هم کیف پول فرستنده و هم کیف پول گیرنده باید ارتقا داده شوند تا بتوانند از segwit استفاده کنند. علاوه‌براین، کیف پول فرستنده باید بداند که کیف پول گیرنده آگاه از segwit است.

شاهد تفکیک‌شده همزمان در کل شبکه پیاده‌سازی نمی‌شود. در عوض، شاهد تفکیک‌شده به عنوان یک ارتقای سازگار با نسخه‌های پیشین پیاده‌سازی می‌شود تا کلاینت‌های قدیمی و جدید بتوانند همزیستی داشته باشند. توسعه‌دهندگان کیف پول به طور مستقل نرم‌افزار کیف پول را ارتقا می‌دهند تا قابلیت‌های segwit را اضافه کنند. انواع پرداخت P2WPKH و P2WSH در مواقعی استفاده می‌شوند که هم فرستنده و هم گیرنده آگاه از segwit باشند. P2PKH و P2SH سنتی برای کیف پول‌های ارتقانیافته به کار خود ادامه خواهند داد که به این منوال، دو سناریوی مهم به جا می‌ماند که در بخش بعدی به آن‌ها پرداخته می‌شود:

- توانایی کیف پول فرستنده‌ای که آگاه از segwit نیست برای پرداخت به کیف پول گیرنده‌ای که می‌تواند تراکنش‌های segwit را پردازش کند.
- توانایی استفاده از کیف پول فرستنده‌ای که آگاه از segwit است برای شناسایی و تمایز قائل شدن میان گیرنده‌های آگاه و ناآگاه از segwit، از طریق آدرس‌هایشان.

ادغام شاهد تفکیک‌شده در درون P2SH

فرض کنیم کیف پول آلیس به segwit ارتقا نیافته است، اما کیف پول باب ارتقایافته و قادر انجام به تراکنش‌های segwit است. آلیس و باب می‌توانند از تراکنش‌های غیر segwit "قدیمی" استفاده کنند. اما باب به احتمال زیاد می‌خواهد از segwit برای کاهش کارمزدهای تراکنش‌ها استفاده کند و از تخفیف موجود در داده‌های شاهد بهره‌مند شود.

در این مورد، کیف پول باب می‌تواند یک آدرس P2SH بسازد که حاوی یک اسکریپت segwit در داخل آن باشد. کیف پول آلیس این موضوع را به عنوان یک آدرس P2SH "عادی" می‌بیند و می‌تواند بدون هیچ گونه آگاهی از segwit به آن پرداخت کند. پس کیف پول باب می‌تواند با بهره‌مندی کامل از segwit و کاهش هزینه‌ی تراکنش، این پرداخت را با یک تراکنش segwit را خرج کند.

هر دو نوع اسکریپت شاهد، P2WPKH و P2WSH، را می‌توان در یک آدرس P2SH ادغام کرد. مورد اول به عنوان P2SH(P2WPKH) و مورد دوم به عنوان P2SH(P2WSH) ذکر می‌شود.

درهم‌سازی کلید عمومی پرداخت به شاهد در درون درهم‌سازی پرداخت به اسکریپت

نخستین شکل از اسکریپت شاهد که بررسی خواهیم کرد P2SH(P2WPKH) است. این یک برنامه‌ی شاهد درهم‌سازی کلید عمومی پرداخت به شاهد است که درون یک اسکریپت درهم‌سازی پرداخت به اسکریپت ادغام شده است، به گونه‌ای که می‌توان از آن با کیف پولی استفاده کرد که از segwit آگاهی ندارد.

کیف پول باب با کلید عمومی باب، یک برنامه‌ی شاهد P2WPKH می‌سازد. سپس این برنامه‌ی شاهد درهم‌سازی می‌شود و درهم‌سازی حاصل به صورت اسکریپت P2SH رمزگذاری می‌شود. اسکریپت P2SH به یک آدرس بیت‌کوین تبدیل می‌شود که با "3" آغاز می‌شود (همان‌طور که در بخش ۷-۳ دیدیم).

کیف پول باب با برنامه‌ی شاهد P2WPKH که پیش‌تر دیدیم، آغاز می‌شود:

```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

برنامه‌ی شاهد P2WPKH شامل نسخه‌ی شاهد و درهم‌سازی کلید عمومی ۲۰ بایتی می‌شود.

کیف پول باب سپس برنامه‌ی شاهد قبلی را درهم‌سازی می‌کند، ابتدا با SHA256، سپس با RIPEMD160 و یک درهم‌سازی ۲۰ بایتی دیگر تولید می‌کند:

```
3e0547268b3b19288b3adef9719ec8659f4b2b0b
```

سپس این درهم‌سازی برنامه‌ی شاهد در یک اسکریپت P2SH ادغام می‌شود:

```
HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL
```

سرانجام، اسکریپت P2SH به یک آدرس بیت‌کوین P2SH تبدیل می‌شود:

```
37Lx99uaGn5avKBxiW26HjedQE3LrDCZru
```

اکنون باب می‌تواند این آدرس را برای مشتری نشان دهد تا بتواند پول قهوه‌اش را بپردازد. کیف پول آلیس می‌تواند به 37Lx99uaGn5avKBxiW26HjedQE3LrDCZru و به هر آدرس بیت‌کوین دیگری پرداخت کند. حتی اگر کیف پول آلیس هیچ گونه حمایتی برای segwit نداشته باشد، پرداختی ایجاد می‌کند که باب با تراکنش segwit قادر است آن را خرج کند.

درهم‌سازی اسکریپت پرداخت به شاهد در درون درهم‌سازی پرداخت به اسکریپت

به طور مشابه، یک برنامه‌ی شاهد P2WSH برای یک اسکریپت multisig یا اسکریپت پیچیده دیگر را می‌توان در درون یک اسکریپت و آدرس P2SH ادغام کرد و به این ترتیب هر کیف پولی می‌تواند پرداخت‌های سازگار با segwit را انجام دهد.

چنان‌که در درهم‌سازی اسکریپت پرداخت به شاهد (P2WSH) دیدیم، شرکت محمد از پرداخت‌های شاهد تفکیک‌شده برای اسکریپت‌های چندگانه استفاده می‌کند. برای این‌که هر کلاینت بتواند به شرکت محمد پول بپردازد، فارغ از اینکه آیا کیف پولش برای segwit به روز شده است یا خیر، کیف پول محمد می‌تواند برنامه‌ی شاهد P2WSH را درون یک اسکریپت P2SH ادغام کند.

نخست، کیف پول محمد برنامه‌ی شاهد P2WSH را که مطابق با اسکریپت چندامضایی است و با SHA256 درهم‌سازی شده، ایجاد می‌کند:

```
0 9592d601848d04b172905e0ddb0adde59f1590f1e553ffc81ddc4b0ed927dd73
```

سپس، برنامه‌ی شاهد خودش با SHA256 و RIPEMD160 hash درهم‌سازی می‌شود و یک درهم‌سازی جدید ۲۰ بایتی تولید می‌کند که در P2SH سنتی استفاده می‌شود:

```
86762607e8fe87c0c37740cddee880988b9455b2
```

آن‌گاه، کیف پول محمد، درهم‌سازی را درون یک اسکریپت P2SH می‌گذارد:

```
HASH160 86762607e8fe87c0c37740cddee880988b9455b2 EQUAL
```

سرانجام، کیف پول، از این اسکریپت یک آدرس بیت‌کوین می‌سازد:

```
3Dwz1MXhM6EfFoJChHCxh1jWHb8GQqRenG
```

اکنون، کلاینت‌های محمد می‌توانند بدون نیاز به پشتیبانی از segwit، به این آدرس پرداخت کنند. پس شرکت محمد می‌تواند با استفاده از ویژگی‌های segwit از جمله هزینه‌های حمل و نقل کمتر، تراکنش‌های segwit را برای خرج کردن این پرداخت‌ها بنا کند.

آدرس شاهد تفکیک‌شده

پس از استقرار segwit در شبکه‌ی بیت کوین، مدتی طول می‌کشد تا کیف پول‌ها ارتقا یابند. از این رو، چنان که در بخش قبل دیدیم، کاملاً محتمل است که دست کم به مدت چند ماه از segwit عمدتاً به شکل ادغام‌شده در P2SH استفاده شود.

اما سرانجام، تقریباً همه‌ی کیف پول‌ها می‌توانند از پرداخت‌های جداگانه پشتیبانی کنند. در آن زمان دیگر نیازی به ادغام segwit در P2SH نخواهد بود. از این رو، احتمال دارد شکل جدیدی از آدرس بیت‌کوین ایجاد شود، شکلی که نشان دهد گیرنده از segwit آگاه است و برنامه‌ی شاهد را به طور مستقیم رمزگذاری می‌کند. چند طرح پیشنهادی برای الگوی آدرس شاهد تفکیک‌شده ارائه شده است، اما هیچ کدام به طور فعال دنبال نشده‌اند.

شناسه‌های تراکنش

یکی از مهم‌ترین فواید شاهد تفکیک‌شده این است که تغییرپذیری تراکنش طرف سوم را حذف می‌کند.

پیش از segwit، طرف‌های سوم می‌توانستند در امضاهای تراکنش‌های خود تغییرات ظریفی بدهند و شناسه‌ی تراکنش (درهم‌سازی) آن‌ها را تغییر دهند بدون این‌که هیچ گونه تغییری در خواص اساسی (ورودی‌ها، خروجی‌ها، مبلغ) داده شود. این برای حملات منع سرویس و همچنین حمله به نرم‌افزارهای کیف پولی که ضعیف نوشته شده بودند (و فرض را بر این می‌گذاشتند که درهم‌سازی‌های تراکنش نشده تاییدنشده، تغییرناپذیرند) ایجاد فرصت می‌کرد.

با معرفی شاهد تفکیک‌شده، تراکنش‌ها دارای دو شناسه، txid و wtxid هستند. شناسه‌ی تراکنش سنتی txid، درهم‌سازیِ SHA256 مضاعفِ تراکنش‌های پی‌درپی، بدون داده‌های شاهد است. تراکنش wtxid، درهم‌سازی SHA256 مضاعف قالبِ پی‌درپی جدید تراکنش با داده‌های شاهد است.

txid سنتی دقیقاً به همان روشی محاسبه می‌شود که با یک تراکنش غیر segwit محاسبه می‌شود. اما از آن‌جا که تراکنش segwit در هر ورودی دارای scriptSigهای خالی است، هیچ بخشی از تراکنش وجود ندارد که توسط طرف سوم قابل تغییر باشد. بنابراین، در یک تراکنش segwit، طرف سوم قادر به تغییر txid نیست، حتی هنگامی که تراکنش تأیید نشده باشد.

wtxid مانند شناسه‌ی بسط‌یافته عمل می‌کند، به این ترتیب که درهم‌سازی، داده‌های شاهد را نیز شامل می‌شود. اگر تراکنشی بدون داده‌های شاهد منتقل شود، wtxid و txid یکسان هستند. توجه دارید که چون wtxid شامل داده‌های شاهد (امضاها) است و از آن‌جا که ممکن است داده‌های شاهد انعطاف‌پذیر باشند، wtxid تا زمان تأیید تراکنش، باید انعطاف‌پذیر تلقی شود. فقط txid یک تراکنش segwit توسط طرف سوم را می‌توان تغییرناپذیر دانست و تنها در صورتی که تمامی ورودی‌های تراکنش، ورودی‌های segwit باشند.

تراکنش‌های شاهد تفکیک‌شده دارای دو شناسه‌اند: txid و wtxid.txid، درهم‌سازی تراکنش بدون داده‌های شاهد و wtxid درهم‌سازی دربرگیرنده‌ی داده‌های شاهد است. txid تراکنشی که در آن همه‌ی ورودی‌ها، ورودی‌های segwit هستند، مستعد ابتلا به تغییرپذیری تراکنش‌های طرف سوم نیست.

الگوریتم جدید امضای شاهد تفکیک‌شده

شاهد تفکیک‌شده معنای ۴ تابع صحت‌سنجی امضا (CHECKSIG، CHECKSIGVERIFY، CHECKMULTISIG و CHECKMULTISIGVERIFY) را اصلاح می‌کند و شیوه‌ی محاسبه درهم‌سازی تعهد تراکنش را تغییر می‌دهد.

امضاها در تراکنش‌های بیت‌کوین روی یک درهم‌سازی تعهد اعمال می‌شوند که از روی داده‌های تراکنش محاسبه می‌شود و بخش‌های خاصی از داده‌های نشانگر تعهد امضاکننده به آن مقادیر را قفل می‌کند. به عنوان مثال، در یک امضای نوع ساده SIGHASH_ALL، درهم‌سازی تعهد، همه ورودی‌ها و خروجی‌ها را در بر می‌گیرد.

متأسفانه، شیوه‌ی محاسبه درهم‌سازی تعهد این امکان را فراهم می‌کرد که یک گره تأیید امضا را بتوان وادار به انجام تعداد قابل توجهی از محاسبات درهم‌سازی کرد. به طور مشخص، اَعمال درهم‌سازی نسبت به تعداد اَعمال امضای موجود در تراکنش به‌صورت $O(n^2)$ افزایش می‌یابد. بنابراین، مهاجم می‌تواند تراکنشی با تعداد بسیار زیادی از اَعمال امضا ایجاد کند و باعث شود کل شبکه‌ی بیت‌کوین مجبور به انجام صدها یا هزاران اَعمال درهم‌سازی برای تأیید تراکنش شود.

Segwit با تغییر در چگونگی محاسبه‌ی درهم‌سازی تعهد، فرصتی برای پرداختن به این مشکل ارائه داد. برای برنامه‌های شاهد segwit نسخه‌ی صفر، تأیید امضا با استفاده از یک الگوریتم درهم‌سازی تعهد بهبودیافته مطابق با BIP-143 رخ می‌دهد.

این الگوریتم جدید به دو هدف مهم دست می‌یابد. در مرحله‌ی نخست، تعداد اَعمال درهم‌سازی با O(n) به مراتب آهسته‌تر نسبت به تعداد اَعمال امضا افزایش می‌یابد و فرصت ایجاد حملات منع سرویس با تراکنش‌های بسیار پیچیده را کاهش می‌دهد. دوم، درهم‌سازی تعهد اکنون شامل ارزش (مبالغ) هر ورودی به عنوان بخشی از تعهد نیز می‌شود. این بدان معناست که امضاکننده می‌تواند بدون نیاز به "واکِشی" و بررسی تراکنش قبلی اشاره‌شده توسط ورودی، به یک مقدار ورودی خاص متعهد شود. در مورد دستگاه‌های آفلاین، مانند کیف پول‌های سخت‌افزاری، این امر ارتباط میان میزبان و کیف پول سخت‌افزاری را بسیار ساده می‌کند و نیاز به جریانی از تراکنش‌های قبلی را برای اعتبارسنجی برطرف می‌کند. کیف پول سخت‌افزاری می‌تواند مقدار ورودی "آن چنان که بیان شده است" توسط یک میزبان غیر قابل اعتماد را بپذیرد. از آن‌جا که چنین امضایی نامعتبر است، اگر آن مقدار ورودی صحیح نباشد، اعتبارسنجی ارزش توسط کیف پول سخت افزاری، پیش از امضای ورودی ضرورتی ندارد.

مشوق‌های اقتصادی برای شاهد تفکیک‌شده

گره‌های استخراج بیت‌کوین و گره‌های کامل، هزینه‌هایی برای منابع مورد استفاده برای پشتیبانی از شبکه‌ی بیت‌کوین و بلاک‌چین به بار می‌آورند. با افزایش حجم تراکنش‌های بیت‌کوین، هزینه منابع (پردازنده، پهنای باند شبکه، فضای دیسک، حافظه) نیز افزایش می‌یابد. این هزینه‌ها برای ماینرها از طریق کارمزدهایی متناسب با اندازه‌ی هر تراکنش (برحسب بایت) جبران می‌شود. هزینه‌ی گره‌های کامل غیراستخراجی جبران نمی‌شود، بنابراین آن‌ها این هزینه‌ها را اعمال می‌کنند زیرا نیاز به اجرای مرجعی دارند که گره‌ای با شاخص کامل را ارزیابی کند، شاید به این دلیل که از این گره برای راه‌اندازی یک کسب‌وکار بیت‌کوینی استفاده می‌کنند.

می‌توان استدلال کرد که بدون هزینه‌های تراکنش، رشد داده‌های بیت‌کوین افزایشی چشمگیر خواهد داشت. هزینه‌ها با هدف هماهنگی نیازهای کاربران بیت‌کوین با باری که تراکنش‌های آن‌ها به شبکه تحمیل می‌کنند، از طریق یک راهکار کشف قیمت مبتنی بر بازار تعیین می‌شوند.

در محاسبه‌ی هزینه‌ها بر اساس اندازه‌ی تراکنش، تمامی داده‌های موجود در تراکنش یکسان قلمداد می‌شوند. اما از دیدگاه گره‌های کامل و ماینرها، بخش‌هایی از یک تراکنش، هزینه‌های بسیار بالاتری را سبب می‌شوند. هر تراکنش افزوده شده به شبکه‌ی بیت‌کوین بر مصرف چهار منبع در گره‌ها تأثیر می‌گذارد:

فضای دیسک

هر تراکنش در بلاک چین ذخیره می‌شود و به کل اندازه‌ی بلاک‌چین اضافه می‌کند. بلاک‌چین روی دیسک ذخیره می‌شود، اما با "هرس کردن" تراکنش‌های قدیمی می‌توان فضای ذخیره‌سازی را بهینه کرد.

پردازنده

هر تراکنش باید اعتبارسنجی شود که این نیاز به زمان پردازنده دارد.

پهنای باند

هر تراکنش حداقل یک بار (از طریق انتشار سیلابی) در سراسر شبکه پخش می‌شود. بدون هیچ گونه بهینه‌سازی در پروتکل انتشار بلاک، تراکنش‌ها دوباره به عنوان بخشی از بلاک منتقل می‌شوند و تأثیر آن بر ظرفیت شبکه دو برابر می‌شود.

حافظه

گره‌هایی که تراکنش‌ها را اعتبارسنجی می‌کنند شاخص UTXO یا کل مجموعه UTXO را در حافظه نگه می‌دارند تا به اعتبارسنجی سرعت بخشند. از آن‌جا که حافظه دست کم ده بار از فضای دیسک گران‌تر است، رشد مجموعه‌ی UTXO به طور نامتناسب هزینه‌ی اجرای گره را بسیار بالا می‌برد.

همان‌طور که از لیست پیداست، همه‌ی بخش‌های تراکنش، تأثیری برابر بر هزینه‌ی اجرای یک گره یا توانایی بیت‌کوین برای تغییر مقیاس به منظور پشتیبانی از تراکنش‌های بیشتر ندارند. پرهزینه‌ترین بخش تراکنش، خروجی‌های تازه ایجاد شده است، زیرا به مجموعه‌ی درون‌حافظه‌ای UTXO افزوده می‌شود. در مقایسه، امضاها (که به عنوان داده‌های شاهد نیز شناخته می‌شوند) کمترین هزینه را به شبکه و هزینه‌ی اجرای گره اضافه می‌کنند، زیرا داده‌های شاهد فقط یک بار اعتبارسنجی می‌شوند و دیگر هرگز دوباره استفاده نمی‌شوند. علاوه‌براین، بلافاصله پس از دریافت یک تراکنش جدید و اعتبارسنجی داده‌های شاهد، گره‌ها می‌توانند آن داده‌های شاهد را دور بیندازند. اگر هزینه‌ها بر اساس اندازه‌ی تراکنش، بدون قائل شدن هیچ تبعیضی میان این دو نوع داده‌ها، محاسبه شوند، در آن صورت مشوق‌های بازاری کارمزدها با هزینه‌های واقعی تحمیل شده توسط یک تراکنش هم‌سو نخواهند بود. در واقع، ساختار فعلی کارمزد رفتار مخالف را تشویق می‌کند، زیرا داده‌های شاهد بزرگترین بخش یک تراکنش هستند.

مشوق‌های ایجادشده توسط هزینه‌ها مهم هستند زیرا بر رفتار کیف پول‌ها تأثیر می‌گذارند. کلیه کیف پول‌ها باید راهبردی را در تراکنش‌ها پیاده‌سازی کنند که چند عامل، مانند حریم خصوصی (کاهش استفاده مجدد از آدرس)، تکه تکه شدن (ایجاد تغییرات سست فراوان) و کارمزدها را مدنظر قرار دهد. اگر کارمزدها به شدت کیف پول را تشویق به حداقل استفاده‌ی ممکن از ورودی‌ها در تراکنش‌ها تشویق کنند، نتیجه می‌تواند انتخاب و تغییر راهبردهای آدرس توسط UTXO شود که سهواً مجموعه‌ی UTXO را متورم می‌کنند.

تراکنش‌ها، UTXO را در ورودی‌های خود مصرف کرده با خروجی‌های خود، UTXO جدید ایجاد می‌کنند. بنابراین، تراکنشی که ورودی بیشتری نسبت به خروجی‌ها داشته باشد، منجر به کاهش در مجموعه‌ی UTXO خواهد شد، در حالی که تراکنشی که دارای خروجی‌های بیشتری نسبت به ورودی‌ها باشد، منجر به افزایش مجموعه UTXO خواهد شد. اکنون تفاوت میان ورودی و خروجی را در نظر می‌گیریم و آن را Net-new-UTXO می‌نامیم. این یک معیار مهم است، زیرا به ما می‌گوید تراکنش روی گران‌ترین منبع شبکه، یعنی مجموعه‌ی درون‌حافظه‌ای UTXO چه تأثیری خواهد گذاشت. تراکنشی با Net-new-UTXO مثبت بر آن بار می‌افزاید. تراکنشی با Net-new-UTXO منفی بار را کاهش می‌دهد. بنابراین، می‌خواهیم تراکنش‌هایی با Net-new-UTXO منفی یا Net-new-UTXO صفر را تشویق کنیم.

اکنون با یک مثال نشان می‌دهیم با محاسبه‌ی کارمزد تراکنش –با و بدون شاهد تفکیک‌شده– چه مشوق‌هایی ایجاد می‌شوند. دو تراکنش متفاوت را در نظر می‌گیریم. تراکنش A، با سه ورودی و دو خروجی که معیار Net-new-UTXOآن 1- است، بدین معنی که یک UTXO بیشتر از آن‌چه ایجاد می‌کند، به مصرف می‌رساند و UTXO را یک واحد کاهش می‌دهد. تراکنش B، با دو ورودی و سه خروجی است معیار Net-new-UTXO آن 1 است، بدین معنی که یک UTXO کمتر از آنچه ایجاد می‌کند، به مصرف می‌رساند و UTXO را یک واحد افزایش می‌دهد و هزینه‌ای بر کل شبکه‌ی بیت‌کوین تحمیل می‌کند. هر دو تراکنش از اسکریپت‌های چندامضایی (دو از سه) استفاده می‌کنند تا نشان دهند اسکریپت‌های پیچیده چگونه اثر شاهد تفکیک‌شده بر کارمزدها را افزایش می‌دهند. تراکنشی با کارمزد ۳۰ ساتوشی به ازای هر بایت و ۷۵٪ تخفیف برای داده‌های شاهد را در نظر بگیرید:

بدون شاهد تفکیک‌شده

کارمزد تراکنش A: ۲۵۷۱۰ ساتوشی

کارمزد تراکنش B: ۱۸۹۹۰ ساتوشی

با شاهد تفکیک‌شده

کارمزد تراکنش A: ۸۱۳۰ ساتوشی

کارمزد تراکنش B: ۱۲۰۴۵ ساتوشی

هنگام اجرای شاهد تفکیک‌شده، هر دو تراکنش کم‌هزینه‌ترند. اما با مقایسه‌ی هزینه‌ها میان دو تراکنش، می‌بینیم که پیش از شاهد تفکیک‌شده، کارمزد تراکنشی با Net-new-UTXO منفی، بیشتر است. پس از شاهد تفکیک‌شده، کارمزدهای تراکنش‌ها با مشوقی برای کمینه‌سازی ایجاد UTXO جدید با جریمه نکردن سهوی تراکنش‌ها با بسیاری از ورودی‌ها همسویی دارد.

بنابراین شاهد تفکیک‌شده دو تأثیر عمده بر کارمزدهای پرداخت شده توسط کاربران بیت‌کوین دارد. نخست، segwit با تخفیف دادن به داده‌های شاهد و افزایش ظرفیت بلاک چین بیت‌کوین، هزینه کلی تراکنش‌ها را کاهش می‌دهد. دوم، تخفیف segwit بر داده‌های شاهد، ناهمسویی مشوق‌هایی را که ممکن است ناخواسته سبب ایجاد تورم بیشتری در مجموعه UTXO شود، تصحیح می‌کند.

پیوست ۵ بیت‌کور

بیت‌کور (Bitcore) مجموعه‌ای از ابزارهای ارائه شده توسط BitPay است. هدف آن تهیه ابزارهایی با کاربرد آسان برای توسعه‌دهندگان بیت‌کوین است. تقریباً تمام کدهای بیت کور به زبان جاوا اسکریپت نوشته شده‌اند. بعضی از ماژول‌ها مشخصاً برای NodeJS نوشته شده‌اند. سرانجام، ماژول "گره" بیت‌کور شامل کد C ++ هسته‌ی بیت‌کوین می‌شود. برای اطلاعات بیشتر لطفاً به https://bitcore.io مراجعه کنید.

فهرست ویژگی‌های بیت‌کوین

- گره کامل بیت‌کوین (bitcore-node)
- کاوشگر بلاک (insight)
- امکانات بلاک، تراکنش و کیف پول (bitcore-lib)
- برقراری ارتباط مستقیم با شبکه P2P بیت‌کوین (bitcore-p2p)
- مولد حافظه آنتروپی (bitcore-mnemonic)
- پروتکل پرداخت (bitcore-payment-protocol)
- صحت‌سنجی و امضای پیام (bitcore-message)
- طرح رمزگذاری با انتگرال‌های بیضوی (bitcore-ecies)
- سرویس کیف پول (bitcore-wallet-service)
- کلاینت کیف پول (bitcore-wallet-client)
- زمین بازی (bitcore-playground)
- یکپارچه‌سازی مستقیم خدمات هسته‌ی بیت‌کوین (bitcore-node)

مثال‌هایی از کتابخانه‌ی بیت‌کور

پیش‌نیازها

- NodeJS >= 4.x؛ از زمین بازی آنلاین ما هم می‌توانید استفاده کنید[1].

درصورت استفاده از NodeJS و node REPL:

```
$ npm install -g bitcore-lib bitcore-p2p
$ NODE_PATH=$(npm list -g | head -1)/node_modules node
```

مثال‌های کیف پول با استفاده از bitcore-lib

ایجاد یک آدرس بیت‌کوین جدید با کلید خصوصی وابسته:

```
> bitcore = require('bitcore-lib')
> privateKey = new bitcore.PrivateKey()
> address = privateKey.toAddress().toString()
```

ایجاد آدرس و کلید خصوصی قطعی سلسله مراتبی:

```
> hdPrivateKey = bitcore.HDPrivateKey()
> hdPublicKey = bitcore.HDPublicKey(hdPrivateKey)
> hdAddress = new bitcore.Address(hdPublicKey.publicKey).toString()
```

ایجاد و امضای تراکنش از یک UTXO:

```
> utxo = {
  txId: transaction id containing an unspent output,
  outputIndex: output indexi e.g. 0,
  address: addressOfUtxo,
  script: bitcore.Script.buildPublicKeyHashOut(addressOfUtxo).toString(),
  satoshis: amount sent to the address
}
> fee = 3000 //set appropriately for conditions on the network
> tx = new bitcore.Transaction()
  .from(utxo)
  .to(address, 35000)
  .fee(fee)
  .enableRBF()
  .sign(privateKeyOfUtxo)
```

جایگزینی آخرین تراکنش در مخزن حافظه (جایگزینی با اجرت):

```
> rbfTx = new Transaction()
  .from(utxo)
  .to(address, 35000)
  .fee(fee*2)
  .enableRBF()
  .sign(privateKeyOfUtxo);
> tx.serialize();
> rbfTx.serialize();
```

انتشار یک تراکنش در شبکه بیت‌کوین (توجه: فقط تراکنش‌های معتبر پخش می‌شوند؛ برای میزبان‌های نظیر به https://bitnodes.21.co/nodes مراجعه کنید):

۱. کد زیر را در فایلی به نام broadcast.js کپی کنید.

۲. متغیرهای tx و rbfTx به ترتیب خروجی‌های tx.serialize() و rbfTx.serialize() هستند.

1. https://bitcore.io/playground

۳. به منظور جایگزینی با اجرت، نظیر باید از گزینه بیت‌کوینی mempoolreplace پشتیبانی کند و آن را برابر با 1 قرار دهد.

۴. فایل گره broadcast.js را اجرا کنید:

```
var p2p = require('bitcore-p2p');
var bitcore = require('bitcore-lib');
var tx = new bitcore.Transaction('output from serialize function');
var rbfTx = new bitcore.Transaction('output from serialize
function');
var host = 'ip address'; //use valid peer listening on tcp 8333
var peer = new p2p.Peer({host: host});
var messages = new p2p.Messages();
peer.on('ready', function() {
  var txs = [messages.Transaction(tx), messages.Transaction(rbfTx)];
  var index = 0;
  var interval = setInterval(function() {
    peer.sendMessage(txs[index++]);
    console.log('tx: ' + index + ' sent');
    if (index === txs.length) {
       clearInterval(interval);
       console.log('disconnecting from peer: ' + host);
       peer.disconnect();
    }
  }, 2000);
});
peer.connect();
```

پیوست ۶ پای‌کوین، Ku و tx

کتابخانه‌ی پای‌کوین (pycoin)[1] که ابتدا توسط ریچارد کیس[2] نوشته و نگهداری شد، کتابخانه‌ای مبتنی بر پایتون است که از دستکاری کلیدها و تراکنش‌های بیت‌کوین و حتی از زبان اسکریپت‌نویسی کافی برای پرداختن به تراکنش‌های غیراستاندارد پشتیبانی می‌کند.

کتابخانه‌ی پای‌کوین از هر دو نسخه‌ی دوم (2.7.x) و سوم (پس از 3.3) پایتون پشتیبانی می‌کند و با ابزارهای خط فرمان سودمند ku و tx همراه است.

ابزار کلیدها (KU)

ابزار خط فرمان ku، حکم یک چاقوی نظامی سویسی را برای دستکاری کلیدها را دارد. این برنامه از کلیدها، WIF و آدرس‌های BIP-32 پشتیبانی می‌کند. چند مثال در زیر آورده شده است.

ایجاد یک کلید BIP-32 با استفاده از منابع آنتروپی پیش فرض GPG و /dev/random:

```
$ ku create

input         : create
network       : Bitcoin
wallet key    : xprv9s21ZrQH143K3LU5ctPZTBnb9kTjA5Su9DcWHvXJe-
miJBsY7VqXUG7hipgdWaU
                m2nhnzdvxJf5KJo9vjP2nABX65c5sFsWsV8oXcbpehtJi
public version : xpub661MyMwAqRbcFpYYiuvZpKjKhnJD-
ZYAkWSY76JvvD7FH4fsG3Nqiov2CfxzxY8
                DGcpfT56AMFeo8M8KPkFMfLUtvwjwb6WPv8rY65L2q8Hz
tree depth    : 0
fingerprint   : 9d9c6092
parent f'print: 00000000
child index   : 0
chain code    :
80574fb260edaa4905bc86c9a47d30c697c50047ed466c0d4a5167f6821e8f3c
private key   : yes
secret exponent :
```

1. http://github.com/richardkiss/pycoin 2. Richard Kiss

```
112471538590155650688604752840386134637231974546906847202389294096567806844862
hex :
f8a8a28b28a916e1043cc0aca52033a18a13cab1638d544006469bc171fddfbe
wif : L5Z54xi6qJusQT42JHA44mfPVZGjyb4XBRWfxAzUWwRiGx1kV4sP
 uncompressed : 5KhoEavGNNH4GHKoy2Ptu4KfdNp4r56L5B5un8FP6RZnbsz5Nmb
public pair x :
764606382405464783648433974782784681018771177678734621270215603682901140160 34
public pair y :
598078796574697741020401202982722077309212917366332477370774067536768257777 01
 x as hex :
a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
 y as hex :
843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
y parity : odd
key pair as sec :
03a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322
  uncompressed :
04a90b3008792432060fa04365941e09a8e4adf928bdbdb9dad41131274e379322843a0f6ed9c0eb1962c74533795406914fe3f1957c5238951f4fe245a4fcd625
hash160 : 9d9c609247174ae323acfc96c852753fe3c8819d
  uncompressed : 8870d869800c9b91ce1eb460f4c60540f87c15d7
Bitcoin address : 1FNNRQ5fSv1wBi5gyfVBs2rkNheMGt86sp
uncompressed : 1DSS5isnH4FsVaLVjeVXewVSpfqktdiQAM
```

ایجاد یک کلید BIP-32 با استفاده از کلمه‌ی رمز:

در این مثال، کلمه‌ی رمز را به آسانی می‌توان حدس زد:

```
$ ku P:foo

input : P:foo
network : Bitcoin
wallet key :
xprv9s21ZrQH143K31AgNK5pyVvW23gHnkBq2wh5aEk6g1s496M8ZMjxncCKZKgb5j
          ZoY5eSJMJ2Vbyvi2hbmQnCuHBujZ2WXGTux1X2k9Krdtq
public version : xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31-tgH6u4DLQWoQayvtS
          VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
tree depth : 0
fingerprint : 5d353a2e
parent f'print : 00000000
child index : 0
chain code :
5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc
private key : yes
secret exponent :
658257305470973057160571604379707902201238642997619089487468358860077939982 75
 hex :
91880b0e3017ba586b735fe7d04f1790f3c46b818a2151fb2def5f14dd2fd9c3
wif : L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

```
uncompressed : 5JvNzA5vXDoKYJdw8SwwLHxUxaWvn9mDea6k1vRPCX7KLUVWa7W
public pair x :
818219827193811040617773492691304190244936166509935893945534043477743
93168191
public pair y :
589942180696054242783207032506897801547850995092776917231263250512004
59038290
 x as hex :
b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 y as hex :
826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
y parity : even
key pair as sec :
02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f
 uncompressed :
04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f

826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52
hash160 : 5d353a2ecdb262477172852d57a3f11de0c19286
 uncompressed : e5bd3a7e6cb62b4c820e51200fb1c148d79e67da
Bitcoin address : 19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
 uncompressed : 1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT
```

دریافت اطلاعات به صورت JSON:

```
$ ku P:foo -P -j
{
  "y_parity": "even",
  "public_pair_y_hex":
"826d8b4d3010aea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "private_key": "no",
  "parent_fingerprint": "00000000",
  "tree_depth": "0",
  "network": "Bitcoin",
  "btc_address_uncompressed": "1MwkRkogzBRMehBntgcq2aJhXCXStJTXHT",
  "key_pair_as_sec_uncompressed":
"04b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f8
26d8b4d3010a
ea16ff4c1c1d3ae68541d9a04df54a2c48cc241c2983544de52",
  "public_pair_x_hex":
"b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f",
  "wallet_key":
"xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWo
QayvtSVYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy",
  "chain_code":
"5eeb1023fd6dd1ae52a005ce0e73420821e1d90e08be980a85e9111fd7646bbc",
  "child_index": "0",
  "hash160_uncompressed": "e5bd3a7e6cb62b4c820e51200fb1c148d79e67da",
  "btc_address": "19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii",
  "fingerprint": "5d353a2e",
  "hash160": "5d353a2ecdb262477172852d57a3f11de0c19286",
  "input": "P:foo",
  "public_pair_x":
"818219827193811040617773492691304190244936166509935893945534043477
4393168191",
  "public_pair_y":
```

```
"589942180696054242783207032506897801547850995092776917231263250512004590382 90",
  "key_pair_as_sec":
"02b4e599dfa44555a4ed38bcfff0071d5af676a86abf123c5b4b4e8e67a0b0b13f"
}
```

کلید BIP-32 عمومی:

```
$ ku -w -P P:foo
xpub661MyMwAqRbcFVF9ULcqLdsEa5WnCCugQAcgNd9iEMQ31tgH6u4DLQWoQayvtS-
VYFvXz2vPPpbXE1qpjoUFidhjFj82pVShWu9curWmb2zy
```

تولید زیرکلید:

```
$ ku -w -s3/2 P:foo
xprv9wTErTSkjVyJa1v4cUTFMFkWMe5eu8ErbQcs9xajn-
sUzCBT7ykHAwdrxvG3g3f6BFk7ms5hHBvmbdutNmyg6iogWKxx6mefEw4M8EroLgKj
```

تولید زیرکلید:

```
$ ku -w -s3/2H P:foo
xprv9wTErTSu5AWGk-
DeUPmqBcbZWX1xq85ZNX9iQRQW9DXwygFp7iRGJo79dsVctcsCHsnZ3XU3DhsuaGZbDh
8iDkBN45k67U
KsJUXM1JfRCdn1
```

:WIF

```
$ ku -W P:foo
L26c3H6jEPVSqAr1usXUp9qtQJw6NHgApq6Ls4ncyqtsvcq2MwKH
```

آدرس:

```
$ ku -a P:foo
19Vqc8uLTfUonmxUEZac7fz1M5c5ZZbAii
```

تولید یک دسته زیرکلید:

```
$ ku P:foo -s 0/0-5 -w
xprv9xWkBDfyBXmZjBG9EiXBpy67KK72fphUp9utJokEBFtjsjiuKUUDF5V3TU8U8cDz
ytqYn-
Sekc8bYuJS8G3bhXxKWB89Ggn2dzLcoJsuEdRK
xprv9xWkBDfyBXmZnzKf3bAGifK593gT7WJZPnYAmvc77gUQ-
Vej5QHckc5Adtwxa28ACmANi9XhCrRvtFqQcUxt8rUgFz3souMiDdWxJDZnQxzx
xprv9xWkBDfyBXmZqdXA8y4SWqfBdy71gSW9sjx9JpCiJEiBwSMQyRxan6srXUPBtj3P
TxQFkZJAi-
woUpmvtrxKZu4zfsnr3pqyy2vthpkwuoVq
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
xprv9xWkBDfyBXmZv2q3N66hhZ8DAcEnQDnXML1J62krJAcf7Xb1HJwuW2VMJQrCofY2
jtFXdiEY8UsR
NJfqK6DAdyZXoMvtaLHyWQx3FS4A9zw
xprv9xWkBDfyBXmZw4jEYXU-
HYc9fT25k9irP87n2RqfJ5bqbjKdT84Mm7Wtc2xmzFuKg7iYf7XFHKkSsaYKWKJbR54b
nyAD9GzjUY-
bAYTtN4ruo
```

تولید آدرس‌های متناظر:

```
$ ku P:foo -s 0/0-5 -a
1MrjE78H1R1rqdFrmkjdHnPUdLCJALbv3x
1AnYyVEcuqeoVzH96zj1eYKwoWfwte2pxu
1GXr1kZfxE1FcK6ZRD5sqqqs5YfvuzA1Lb
```

```
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
1Cz2rTLjRM6pMnxPNrRKp9ZSvRtj5dDUML
1WstdwPnU6HEUPme1DQayN9nm6j7nDVEM
```

تولید WIFهای متناظر:

```
$ ku P:foo -s 0/0-5 -W
L5a4iE5k9gcJKGqX3FWmxzBYQc29PvZ6pgBaePLVqT5YByEnBomx
Kyjgne6GZwPGB6G6kJEhoPbmyjMP7D5d3zRbHVjwcq4iQXD9QqKQ
L4B3ygQxK6zH2NQGxLDee2H9v4Lvwg14cLJW7QwWPzCtKHdWMaQz
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
L2oD6vA4TUyqPF8QG4vhUFSgwCyuuvFZ3v8SKHYFDwkbM765Nrfd
KzChTbc3kZFxUSJ3Kt54cxsogeFAD9CCM4zGB22si8nfKcThQn8C
```

بررسی این‌که آیا با انتخاب یک رشته‌ی BIP32 (رشته‌ای متناظر با زیرکلید 0/3)، جواب می‌دهد یا خیر:

```
$ ku -W
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
L2L2PZdorybUqkPjrmhem4Ax5EJvP7ijmxbNoQKnmTDMrqemY8UF
$ ku -a
xprv9xWkBDfyBXmZsA85GyWj9uYPyoQv826YAadKWMaaEosNrFBKgj2TqWuiWY3zuq-
xYGpHfv9cnGj5P7e8EskpzKL1Y8Gk9aX6QbryA5raK73p
116AXZc4bDVQrqmcinzu4aaPdrYqvuiBEK
```

بله، آشنا به نظر می‌رسد.

```
$ ku 1

input : 1
network : Bitcoin
secret exponent : 1
hex : 1
wif : KwDiBf89QgGbjEhKnhXJuH7LrciVrZi3qYjgd9M7rFU73sVHnoWn
uncompressed : 5HpHagT65TZzG1PH3CSu63k8DbpvD8s5ip4nEB3kEsreAnchuDf
public pair x :
55066263022277343669578718895168534326250603453777594175500187360389
116729240
public pair y :
32670510020758816978083085130507043184471273380659243275938904335757
337482424
 x as hex :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
 y parity : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
uncompressed :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
 uncompressed : 1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm
```

نسخه‌ی لایت‌کوین:

```
$ ku -nL 1

input : 1
network : Litecoin
secret exponent : 1
 hex : 1
wif : T33ydQRKp4FCW5LCLLUB7deioUMoveiwekdwUwyfRDeGZm76aUjV
 uncompressed : 6u823ozcyt2rjPH8Z2ErsSXJB5PPQwK7VVTwwN4mxLBFrao69XQ
public pair x :
550662630222773436695787188951685343262506034537775941755001873603891
16729240
public pair y :
326705100207588169780830851305070431844712733806592432759389043357573
37482424
 x as hex :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity : even
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
uncompressed :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798

483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
 uncompressed : 91b24bf9f5288532960ac687abb035127b1d28a5
Litecoin address : LVuDpNCSSj6pQ7t9Pv6d6sUkLKoqDEVUnJ
 uncompressed : LYWKqJhtPeGyBAw7WC8R3F7ovxtzAiubdM
```

WIF دوژکوین:

```
$ ku -nD -W 1
QNcdLVw8fHkixm6NNyN6nVwxKek4u7qrioRbQmjxac5TVoTtZuot
```

از زوج عمومی (روی Testnet) داریم:

```
$ ku -nT
550662630222773436695787188951685343262506034537775941755001873603891
16729240,even
input :
550662630222773436695787188951685343262506034537775941755001873603
89116729240,even
network : Bitcoin testnet
public pair x :
550662630222773436695787188951685343262506034537775941755001873603891
16729240
public pair y :
326705100207588169780830851305070431844712733806592432759389043357573
37482424
 x as hex :
79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 y as hex :
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
y parity : even
```

```
key pair as sec :
0279be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
 uncompressed :
0479be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
uncompressed : 91b24bf9f5288532960ac687abb035127b1d28a5
Bitcoin testnet address : mrCDrCybB6J1vRfbwM5hemdJz73FwDBC8r
 uncompressed : mtoKs9V381UAhUia3d7Vb9GNak8Qvmcsme
```

از hash160 داریم:

```
$ ku 751e76e8199196d454941c45d1b3a323f1433bd6

input : 751e76e8199196d454941c45d1b3a323f1433bd6
network : Bitcoin
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
Bitcoin address : 1BgGZ9tcN4rm9KBzDn7KprQz87SZ26SAMH
```

به عنوان آدرس دوژکوین[1]:

```
$ ku -nD 751e76e8199196d454941c45d1b3a323f1433bd6
input : 751e76e8199196d454941c45d1b3a323f1433bd6
network : Dogecoin
hash160 : 751e76e8199196d454941c45d1b3a323f1433bd6
Dogecoin address : DFpN6QqFfUm3gKNaxN6tNcab1FArL9cZLE
```

ابزار تراکنش‌ها (TX)

ابزار خط فرمان tx، تراکنش‌ها را به‌صورتی نمایش می‌دهد که برای انسان قابل خواندن باشد، تراکنش‌های پایه را از حافظه نهان تراکنش‌های پای‌کوین یا از سرویس‌های وب، واکِشی می‌کند (در حال حاضر، block-chain.info، blockcypher.com، blockr.io و chain.so پشتیبانی می‌شوند)، تراکنش‌ها را در هم ادغام می‌کند، ورودی‌ها یا خروجی‌ها را اضافه یا حذف می‌کند، و تراکنش‌ها را امضا می‌کند.

در ادامه چند مثال آورده خواهد شد:

به تراکنش معروف "پیتزا" نگاهی می‌اندازیم:

```
$ tx
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: consider setting environment variable
PYCOIN_CACHE_DIR=~/.pycoin_cache
to cache transactions fetched via web services
warning: no service providers found for get_tx; consider setting
environment
variable PYCOIN_BTC_PROVIDERS
usage: tx [-h] [-t TRANSACTION_VERSION] [-l LOCK_TIME] [-n NETWORK] [-a]
              [-i address] [-f path-to-private-keys] [-g GPG_ARGUMENT]
              [--remove-tx-in tx_in_index_to_delete]
              [--remove-tx-out tx_out_index_to_delete]
              [-F transaction-fee] [-u]
```

[1] Dogecoin

```
                [-b BITCOIND_URL] [-o path-to-output-file]
                argument [argument ... ]
tx: error: can't find Tx with id
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
```

اشتباه! سرویس‌های وب را پیکربندی نکرده بودیم. اکنون این کار را انجام می‌دهیم:

```
$ PYCOIN_CACHE_DIR=~/.pycoin_cache
$ PYCOIN_BTC_PROVIDERS="block.io blockchain.info blockexplorer.com"
$ export PYCOIN_CACHE_DIR PYCOIN_BTC_PROVIDERS
```

این عمل به صورت خودکار انجام نمی‌شود، بنابراین یک ابزار خط فرمان نمی‌تواند اطلاعات خصوصی بالقوه مربوط به تراکنش‌های موردنظر شما را به یک وب‌سایت طرف سوم نشت دهد. اگر برای شما اهمیتی ندارد، می‌توانید این خطوط را در profile. خود قرار دهید.

اکنون دوباره امتحان می‌کنیم:

```
$ tx 9d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a 159 bytes
TxIn count: 1; TxOut count: 1
Lock time: 0 (valid anytime)
Input:
0: (unknown) from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
Output:
 0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total output 10000000.00000 mBTC
including unspents in hex dump since transaction not fully signed
010000000141045e0ab2b0b82cde-
faf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100
a7f26eda8749
31999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e919923
8eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e80000
001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000
** can't validate transaction as source transactions missing
```

خط آخر از این رو ظاهر می‌شود که برای اعتبارسنجی امضای تراکنش‌ها، از نظر فنی به تراکنش‌های مبدأ نیاز دارید. بنابراین، را با افزودن اطلاعات منبع، a- را به تراکنش‌ها اضافه می‌کنیم:

```
$ tx -a 49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a
warning: transaction fees recommendations casually calculated and
estimates may
be incorrect
warning: transaction fee lower than (casually calculated) expected
value of 0.1
mBTC, transaction might not propogate
Version: 1 tx hash
49d2adb6e476fa46d8357babf78b1b501fd39e177ac7833124b3f67b17c40c2a 159 bytes
TxIn count: 1; TxOut count: 1
```

```
Lock time: 0 (valid anytime)
Input:
 0: 17WFx2GQZUmh6Up2NDNCEDk3deYomdNCfk from
1e133f7de73ac7d074e2746a3d6717dfc99ecaa8e9f9fade2cb8b0b20a5e0441:0
10000000.00000 mBTC sig ok
Output:
 0: 1CZDM6oTttND6WPdt3D6bydo7DYKzd9Qik receives 10000000.00000 mBTC
Total input 10000000.00000 mBTC
Total output 10000000.00000 mBTC
Total fees 0.00000 mBTC
010000000141045e0ab2b0b82cde-
faf9e9a8ca9ec9df17673d6a74e274d0c73ae77d3f131e000000004a493046022100
a7f26eda8749
31999c90f87f01ff1ffc76bcd058fe16137e0e63fdb6a35c2d78022100a61e919923
8eb73f07c8f2
09504c84b80f03e30ed8169edd44f80ed17ddf451901ffffffff010010a5d4e80000
001976a9147e
c1003336542cae8bded8909cdd6b5e48ba0ab688ac00000000
all incoming transaction values validated
```

اکنون، به خروجی‌های خرج نشده برای آدرسی مشخص (UTXO) نگاهی می‌اندازیم. در بلاک شماره ۱، تراکنش پایگاه سکه با 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX را می‌بینیم. برای یافتن کلیه سکه‌های موجود در این آدرس از fetch_unspent استفاده می‌کنیم:

```
$ fetch_unspent 12c6DSiU4Rq3P4ZxziKxzrL5LmMBrzjrJX
a3a6f902a51a2cbebede144e48a88c05e608c2cce28024041a5b9874013a1e2a/
0/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/333000
cea36d008badf5c7866894b191d3239de9582d89b6b452b596f1f1b76347f8cb/
31/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
065ef6b1463f552f675622a5d1fd2c08d6324b4402049f68e767a719e2049e8d/
86/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/10000
a66dddd42f9f2491d3c336ce5527d45cc5c2163aaed3158f81dc054447f447a2/0/7
6a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/10000
ffd901679de65d4398de90cefe68d2c3ef073c41f7e8dbec2fb5cd75fe71dfe7/0/7
6a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/100
d658ab87cc053b8dbcfd4aa2717fd23cc3edfe90ec75351fadd6a0f7993b461d/
5/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/911
36ebe0ca3237002acb12e1474a3859bde0ac84b419ec4ae373e63363ebef731c/
1/76a914119b098e2e980a229e139a9ed01a469e518e6f2688ac/100000
fd87f9adebb17f4ebb1673da76ff48ad29e64b7afa02fda0f2c14e43d220fe24/0/7
6a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1
dfdf0b375a987f17056e5e919ee6eadd87dad36c09c4016d4a03cea15e5c05e3/1/7
6a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
cb2679bfd0a557b2dc0d8a6116822f3fcbe281ca3f3e18d3855aa7ea378fa373/0/7
6a914119b098
e2e980a229e139a9ed01a469e518e6f2688ac/1337
d6be34ccf6edddc3cf69842dce99fe503bf632ba2c2adb0f95c63f6706ae0c52/1/7
6a914119b098
```

```
e2e980a229e139a9ed01a469e518e6f2688ac/2000000
0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098/0/4
10496b538e85
3519c726a2c91e61ec11600ae1390813a627c66fb8be7947be63c52da7589379515d
4e0a604f8141
781e62294721166bf621e73a82cbf2342c858eeac/5000000000
```

پیوست ۷ فرمان‌های کاوشگر بیت‌کوین (bx)

کاوشگر بیت‌کوین (bx)[1] یک ابزار خط فرمان است که انواع مختلفی از فرمان‌های مربوط به مدیریت کلیدها و ساخت تراکنش‌ها را ارائه می‌دهد. این ابزار، بخشی از کتابخانه بیت‌کوین (libbitcoin) است.

```
Usage: bx COMMAND [--help]
Info: The bx commands are:
address-decode
address-embed
address-encode
address-validate
base16-decode
base16-encode
base58-decode
base58-encode
base58check-decode
base58check-encode
base64-decode
base64-encode
bitcoin160
bitcoin256
btc-to-satoshi
ec-add
ec-add-secrets
ec-multiply
ec-multiply-secrets
ec-new
ec-to-address
ec-to-public
ec-to-wif
fetch-balance
fetch-header
fetch-height

fetch-history
fetch-stealth
```

1. Bitcoin Explorer

```
fetch-tx
fetch-tx-index
hd-new
hd-private
hd-public
hd-to-address
hd-to-ec
hd-to-public
hd-to-wif
help
input-set
input-sign
input-validate
message-sign
message-validate
mnemonic-decode
mnemonic-encode
ripemd160
satoshi-to-btc
script-decode
script-encode
script-to-address
seed
send-tx
send-tx-node
send-tx-p2p
settings
sha160
sha256
sha512
stealth-decode
stealth-encode
stealth-public
stealth-secret
stealth-shared
tx-decode
tx-encode
uri-decode
uri-encode
validate-tx
watch-address
wif-to-ec
wif-to-public
wrap-decode
wrap-encode
```

برای اطلاعات بیشتر، صفحه‌ی اصلی کاوشگر بیت‌کوین[1] و مستندات کاربری کاوشگر بیت‌کوین[2] را ببینید.

1. https://github.com/libbitcoin/libbitcoin-explorer
2. https://github.com/libbitcoin/libbitcoin-explorer/wiki

مثال‌هایی از خط فرمان bx

اکنون چند نمونه از کاربرد فرمان‌های کاوشگر بیت‌کوین را برای آزمایش کلیدها و آدرس‌ها مرور می‌کنیم.

با استفاده از فرمان seed یک مقدار "بذر" تصادفی را ایجاد کنید که از مولد اعداد تصادفی سیستم‌عامل استفاده می‌کند. بذر را به فرمان ec-new مسترد کنید تا یک کلید خصوصی جدید تولید کند. خروجی استاندارد را در فایل private_key ذخیره می‌کنیم:

```
$ bx seed | bx ec-new > private_key
$ cat private_key
73096ed11ab9f1db6135857958ece7d73ea7c30862145bcc4bbc7649075de474
```

اکنون با استفاده از فرمان ec-to-public، کلید عمومی را از آن کلید خصوصی تولید کنید. فایل priv_key را به ورودی استاندارد مسترد می‌کنیم و خروجی استاندارد این فرمان را در یک فایل public_key جدید ذخیره می‌کنیم:

```
$ bx ec-to-public < private_key > public_key
$ cat public_key
02fca46a6006a62dfdd2dbb2149359d0d97a04f430f12a7626dd409256c12be500
```

می‌توانیم public_key را با استفاده از ec-to-public به عنوان یک آدرس دوباره فرمت کنیم و public_key را به درون ورودی استاندارد مسترد کنیم.

```
$ bx ec-to-address < public_key
17re1S4Q8ZHyCP8Kw7xQad1Lr6XUzWUnkG
```

کلیدهایی که به این روش تولید می‌شوند، یک کیف پول غیرقطعی نوع صفر ایجاد می‌کنند. این بدان معناست که هر کلید از یک بذر مستقل تولید می‌شود. فرمان‌های کاوشگر بیت‌کوین همچنین مطابق با BIP-32 می‌توانند کلیدها را به صورت قطعی تولید کنند. در این حالت، یک کلید "اصلی" از یک بذر ایجاد می‌شود و سپس به طور قطعی بسط داده می‌شود تا درختی از زیرکلیدها تولید شود و در نتیجه یک کیف پول قطعی نوع ۲ ایجاد می‌شود. نخست، از فرمان‌های seed و hd-new برای تولید یک کلید اصلی استفاده می‌کنیم که به عنوان اساسی برای استخراج سلسله مراتب کلیدها استفاده می‌شود:

```
$ bx seed > seed
$ cat seed
eb68ee9f3df6bd4441a9feadec179ff1
$ bx hd-new < seed > master
$ cat master
xprv9s21ZrQH143K2BEhMYpNQoUvAgiEjArAVaZaCTgsaGe6LsAnwu-
beiTcDzd23mAoyizm9cApe51gNfLMkBqkYoWWMCRwzfuJk8RwF1SVEpAQ
```

اکنون از فرمان hd-private برای تولید کلید "حساب" تقویت شده و توالی دو کلید خصوصی در حساب استفاده می‌کنیم:

```
$ bx hd-private --hard < master > account
$ cat account
xprv9vkDLt81dTKjwHB8fsVB5QK8cGnzveChzSrtCfvu3aMWvQaThp59ueu-
fuyQ8Qi3qpjk4aKsbmbfxwcgS8PYbgoR2NWHeLyvg4DhoEE68A1n
```

```
$ bx hd-private --index 0 < account
xprv9xHfb6w1vX9xgZyPNXVgAhPxSsEkeRcPHEUV5iJcVEsuUEACvR3NRY3fpGhcnBiD
bvG4LgndirD-
sia1e9F3DWPkX7Tp1V1u97HKG1FJwUpU
$ bx hd-private --index 1 < account
xprv9xHfb6w1vX9xjc8XbN4GN86jzNAZ6xHEqYxzbLB4fzHFd6VqCLPGRZFsdjsuMVER
adbgDb-
ziCRJru9n6tzEWrASVpEdrZrFidt1RDfn4yA3
```

اکنون از فرمان hd-public برای تولید توالی متناظری از دو کلید عمومی استفاده می‌کنیم:

```
$ bx hd-public --index 0 < account
xpub6BH1zcTuk-
tiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYP
FJ3vezHz5wza
SW4FiGrseNDR4LKqTy
$ bx hd-public --index 1 < account
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWy-
MaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQDhohEcDFTEYMvYzwRD7Juf8
```

کلیدهای عمومی را همچنین می‌توان از کلیدهای خصوصی متناظر با استفاده از فرمان hd-to-public به دست آورد:

```
$ bx hd-private --index 0 < account | bx hd-to-public
xpub6BH1zcTuk-
tiFu43rUZ2gXqLgzu5F3tLEeTQ5t6iE3aQtM2VMTxMcyLN9fYHiGhGpQe9QQYmqL2eYP
FJ3vezHz5wza
SW4FiGrseNDR4LKqTy

$ bx hd-private --index 1 < account | bx hd-to-public
xpub6BH1zcTuktiFx6CzhPbGjG3UYQ13WR16CmtbPiagEKpEVtpyjshWy-
MaMV1cn7nUPUkgQHPVXJVqsrA8xWbGQDhohEcDFTEYMvYzwRD7Juf8
```

می‌توانیم تعداد بی‌شماری از کلیدها را در یک زنجیره‌ی قطعی تولید کنیم که همگی از یک بذر واحد به دست می‌آیند. این تکنیک در بسیاری از برنامه‌های کیف پول برای تولید کلیدهایی مورد استفاده قرار می‌گیرد که بتوان با یک مقدار بذر واحد از آن‌ها نسخه پشتیبان تهیه و بازیابی کرد. این کار ساده‌تر از نیاز به تهیه‌ی نسخه پشتیبان از کیف پول با تمام کلیدهای تولید شده به طور تصادفی با هر بار ایجاد کلید جدید است.

این بذر را می‌توان با استفاده از فرمان mnemonic-decode رمزگذاری کرد:

```
$ bx hd-mnemonic < seed > words
adore repeat vision worst especially veil inch woman cast recall
dwell appreci-
ate
```

سپس می‌توان بذر را با استفاده از فرمان mnemonic-decode رمزگشایی کرد:

```
$ bx mnemonic-decode < words
eb68ee9f3df6bd4441a9feadec179ff1
```

این شیوه رمزگذاری کمک می‌کند بذر راحت‌تر ثبت شود و به یاد آوردن آن هم آسان‌تر می‌شود.