وایت پیپر بیت کوین به همراه ترجمه فارسی

این ترجمه به کمک هوشواره و توسط تیم مدرسه اقتصاد معین انجام شده

متن اصلی وایت پیپر بیت کوین در آخر فایل پیوست شده است.

ترجمه فارسی وایت پیپر بیت کوین:

بیت‌کوین: یک سیستم نقدی الکترونیکی همتا به همتا

ساتوشی ناکاموتو

satoshin@gmx.com

www.bitcoin.org

چکیده:

یک نسخه کاملاً همتا به همتا از پول الکترونیکی این امکان را فراهم می‌کند که پرداخت‌های آنلاین مستقیماً از یک طرف به طرف دیگر ارسال شوند، بدون نیاز به عبور از یک نهاد مالی. امضاهای دیجیتال بخشی از این راه‌حل را فراهم می‌کنند، اما اگر همچنان نیاز به یک واسطه‌ی مورد اعتماد برای جلوگیری از دوبار خرج کردن باشد، مزایای اصلی از بین می‌رود. ما راه‌حلی برای مشکل دوبار خرج کردن پیشنهاد می‌کنیم که از یک شبکه‌ی همتا به همتا استفاده می‌کند. این شبکه با هش کردن تراکنش‌ها و قرار دادن آن‌ها در یک زنجیره‌ی مداوم از اثبات کار مبتنی بر هش، به آن‌ها مهر زمانی می‌زند. این زنجیره یک سابقه‌ی تغییرناپذیر ایجاد می‌کند که فقط با بازانجام اثبات کار قابل تغییر است. طولانی‌ترین زنجیره نه تنها اثباتی از ترتیب رخدادهاست، بلکه نشان می‌دهد که از سوی بزرگ‌ترین مجموعه‌ی توان پردازشی ایجاد شده است. تا زمانی که اکثریت توان پردازشی در اختیار گره‌هایی باشد که قصد حمله به شبکه را ندارند، آن‌ها طولانی‌ترین زنجیره را تولید کرده و از مهاجمان پیشی می‌گیرند. ساختار مورد نیاز شبکه بسیار ساده است. پیام‌ها به صورت تلاش حداکثری (best effort) پخش می‌شوند و گره‌ها می‌توانند در هر زمانی از شبکه خارج شده یا دوباره به آن بپیوندند و طولانی‌ترین زنجیره‌ی اثبات کار را به عنوان سند اتفاقات رخ‌داده در زمان غیبت خود بپذیرند.

مقدمه

تجارت در اینترنت تقریباً به‌طور کامل به نهادهای مالی متکی شده است که به عنوان واسطه‌های مورد اعتماد برای پردازش پرداخت‌های الکترونیکی عمل می‌کنند. این سیستم با وجود کارآمدی نسبی در بسیاری از تراکنش‌ها، همچنان از ضعف‌های ذاتی مدل مبتنی بر اعتماد رنج می‌برد. تراکنش‌هایی که به‌طور کامل غیرقابل برگشت باشند، واقعاً امکان‌پذیر نیستند، زیرا نهادهای مالی ناگزیرند در صورت بروز اختلاف، نقش میانجی را ایفا کنند.

هزینه‌ی این میانجی‌گری باعث افزایش هزینه‌های تراکنش می‌شود، اندازه‌ی حداقل تراکنش‌های عملی را محدود می‌کند و امکان انجام تراکنش‌های کوچک و روزمره را از بین می‌برد. همچنین، ناتوانی در انجام پرداخت‌های غیرقابل برگشت برای خدمات غیرقابل بازگشت، هزینه‌ای گسترده‌تر در پی دارد.

با امکان برگشت تراکنش، نیاز به اعتماد گسترش می‌یابد. فروشندگان باید نسبت به خریداران خود محتاط باشند و اطلاعات بیشتری از آن‌ها بخواهند. درصدی از تقلب نیز به عنوان امری اجتناب‌ناپذیر پذیرفته می‌شود.

در تعاملات حضوری با پول نقد فیزیکی، می‌توان این هزینه‌ها و ابهامات پرداخت را کنار زد، اما در فضای ارتباطی آنلاین، هیچ مکانیزمی برای پرداخت بدون واسطه‌ی مورد اعتماد وجود ندارد.

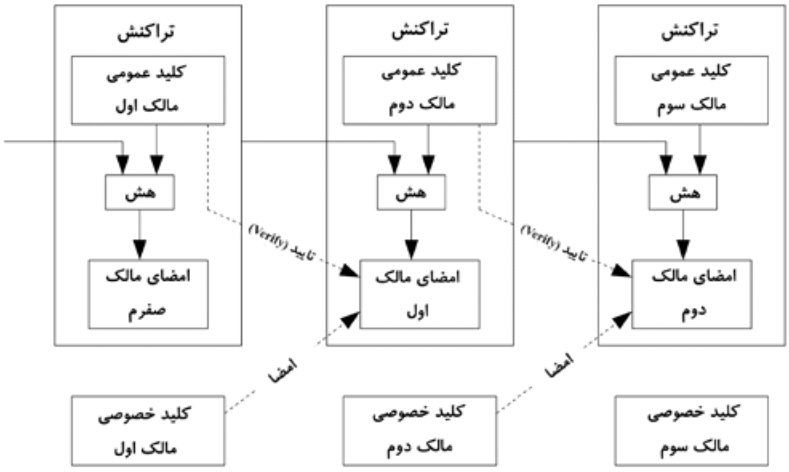
آنچه نیاز است، یک سیستم پرداخت الکترونیکی مبتنی بر اثبات رمزنگاری‌شده است، نه اعتماد، تا دو طرف مایل به انجام تراکنش بتوانند مستقیماً و بدون نیاز به شخص ثالث مورد اعتماد با یکدیگر معامله کنند.

تراکنش‌هایی که به‌لحاظ محاسباتی برگشت‌ناپذیر باشند، فروشنده را از تقلب محافظت می‌کنند و مکانیزم‌های امانی (escrow) ساده‌ای نیز می‌توان برای محافظت از خریدار پیاده‌سازی کرد.

در این مقاله، ما راه‌حلی برای مشکل دوبار خرج کردن ارائه می‌دهیم که از یک سرور زمان‌سنج توزیع‌شده‌ی همتا به همتا استفاده می‌کند تا اثبات محاسباتی ترتیب زمانی تراکنش‌ها را تولید کند. این سیستم تا زمانی که گره‌های صادق در مجموع کنترل توان پردازشی بیشتری نسبت به هر گروه هماهنگ‌شده‌ای از مهاجمان داشته باشند، امن خواهد بود.

تأیید تراکنش‌ها

ما یک "سکه‌ی الکترونیکی" را به صورت زنجیره‌ای از امضاهای دیجیتال تعریف می‌کنیم. هر مالک، سکه را با امضای دیجیتالی یک هش از تراکنش قبلی به همراه کلید عمومی مالک بعدی، به مالک بعدی منتقل می‌کند و این اطلاعات را به انتهای زنجیره‌ی سکه اضافه می‌کند. دریافت‌کننده می‌تواند با بررسی امضاها، زنجیره‌ی مالکیت را تأیید کند.



اما مشکل اینجاست که دریافت‌کننده نمی‌تواند مطمئن باشد که یکی از مالکان قبلی سکه را دوباره خرج نکرده باشد. راه‌حل متداول برای این مشکل، معرفی یک مرجع مرکزی مورد اعتماد (مثل خزانه‌داری یا صادرکننده) است که همه‌ی تراکنش‌ها را برای جلوگیری از دوبار خرج کردن بررسی می‌کند. در این مدل، پس از هر تراکنش، سکه باید به مرجع مرکزی بازگردانده شود تا سکه‌ی جدیدی صادر شود، و تنها سکه‌هایی که مستقیماً از این مرجع صادر شده‌اند قابل‌اعتماد در برابر دوبار خرج کردن هستند.

مشکل این روش آن است که کل نظام پولی وابسته به شرکتی است که مرجع را اداره می‌کند و تمام تراکنش‌ها باید از طریق آن انجام شوند؛ درست مانند یک بانک. ما به روشی نیاز داریم که دریافت‌کننده بتواند بداند مالکان قبلی سکه، هیچ تراکنش قبلی‌ای را امضا نکرده‌اند. برای اهداف ما، فقط اولین تراکنش معتبر است و اهمیتی به تلاش‌های بعدی برای دوبار خرج کردن نمی‌دهیم.

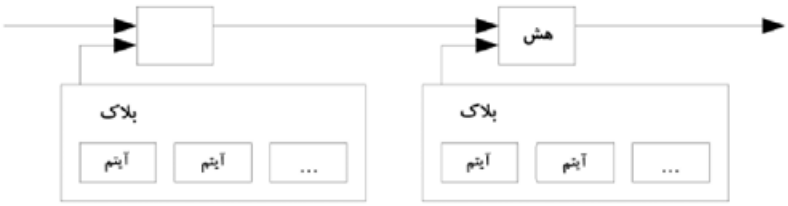
تنها راه برای اطمینان از عدم وجود یک تراکنش، این است که از تمام تراکنش‌ها اطلاع داشته باشیم. در مدل مبتنی بر مرجع مرکزی، این مرجع از تمام تراکنش‌ها آگاه بود و تعیین می‌کرد کدام زودتر رسیده است. برای دستیابی به این هدف

بدون نیاز به واسطه‌ی مورد اعتماد، باید تراکنش‌ها به‌طور عمومی اعلام شوند، و سیستمی نیاز است تا شرکت‌کنندگان روی یک تاریخچه‌ی واحد از ترتیب دریافت تراکنش‌ها به توافق برسند. دریافت‌کننده باید مدرکی داشته باشد که در زمان انجام تراکنش، اکثریت گره‌ها بر این موضوع توافق داشته‌اند که آن تراکنش، اولین تراکنش ثبت‌شده بوده است.

سرور زمان‌سنج

راه‌حلی که ما پیشنهاد می‌کنیم با یک سرور زمان‌سنج آغاز می‌شود. سرور زمان‌سنج با گرفتن هش از یک بلوک شامل مجموعه‌ای از داده‌هایی که باید زمان‌گذاری شوند، و انتشار گسترده‌ی آن هش (مثلاً در روزنامه یا در یک پست در Usenet) عمل می‌کند.

این زمان‌سنجی ثابت می‌کند که داده‌ها در آن زمان مشخص حتماً وجود داشته‌اند، چرا که بدون وجود آن‌ها، تولید هش ممکن نبود. هر زمان‌سنج شامل هش زمان‌سنج قبلی نیز هست، و به این ترتیب یک زنجیره ایجاد می‌شود که هر زمان‌سنج جدید، زمان‌سنج‌های قبلی را تقویت و تأیید می‌کند.



اثبات کار

برای پیاده‌سازی یک سرور زمان‌سنج توزیع‌شده به‌صورت همتا به همتا، به جای استفاده از روزنامه یا پست‌های Usenet، باید از سیستمی بر پایه‌ی «اثبات کار» مشابه با Hashcash ساخته‌ی آدام بک استفاده کنیم. در این سیستم، اثبات کار شامل جست‌وجوی مقداری است که وقتی با الگوریتمی مانند SHA-256 هش می‌شود، هش به تعداد مشخصی بیت صفر در ابتدای خود برسد. میانگین کاری که برای یافتن چنین مقداری نیاز است، به‌صورت نمایی با تعداد بیت‌های صفر مورد نیاز افزایش می‌یابد، در حالی که اعتبار آن تنها با یک هش قابل بررسی است.

در شبکه‌ی زمان‌سنج ما، اثبات کار با افزایش دادن یک مقدار متغیر (nonce) در بلوک انجام می‌شود تا زمانی که مقدار نهایی هش بلوک، شرایط مورد نظر از نظر تعداد بیت صفر را داشته باشد. پس از صرف توان پردازشی برای رسیدن به این نتیجه، تغییر دادن بلوک بدون بازانجام اثبات کار امکان‌پذیر نیست. همچنین، با اضافه شدن بلوک‌های جدید به انتهای زنجیره، تغییر یک بلوک گذشته نیازمند بازانجام اثبات کار تمام بلوک‌های بعد از آن خواهد بود.

اثبات کار همچنین مشکل نمایندگی در تصمیم‌گیری‌های اکثریتی را حل می‌کند. اگر اکثریت بر اساس «هر IP یک رأی» تعیین می‌شد، هر کسی که توانایی اختصاص تعداد زیادی IP داشت می‌توانست سیستم را دست‌کاری کند. اما در مدل اثبات کار، عملاً معادل «هر پردازنده یک رأی» است. تصمیم اکثریت توسط طولانی‌ترین زنجیره‌ای نمایندگی می‌شود که بیشترین تلاش محاسباتی صرف آن شده است. اگر اکثریت توان پردازشی در اختیار گره‌های صادق باشد، زنجیره‌ی صادق سریع‌تر رشد می‌کند و از هر زنجیره‌ی رقیب پیشی می‌گیرد.

برای تغییر یک بلوک در گذشته، مهاجم باید اثبات کار آن بلوک و تمام بلوک‌های پس از آن را مجدداً انجام دهد و سپس به زنجیره‌ی صادق برسد و از آن عبور کند. در ادامه نشان خواهیم داد که احتمال موفقیت مهاجمی که سرعت کمتری دارد، با اضافه شدن بلوک‌های جدید به‌صورت نمایی کاهش می‌یابد. برای سازگاری با افزایش سرعت سخت‌افزار و تغییر علاقه به اجرای گره‌ها در طول زمان، سختی اثبات کار بر اساس میانگین متحرک تنظیم می‌شود تا میانگین تعداد بلوک‌های تولیدشده در ساعت ثابت بماند. اگر تولید بلوک‌ها خیلی سریع شود، سختی افزایش می‌یابد.

شبکه(Network)

مراحل اجرای شبکه به این صورت است:

1. تراکنش‌های جدید به تمام گره‌ها منتشر می‌شود.
2. هر گره تراکنش‌های جدید را جمع‌آوری کرده و آن‌ها را در یک بلوک قرار می‌دهد.

3. هر گره برای بلوک خود به دنبال یافتن یک اثبات کار دشوار می‌گردد.
4. زمانی که گره‌ای اثبات کار را پیدا می‌کند، بلوک را به همه‌ی گره‌ها ارسال می‌کند.
5. گره‌ها فقط زمانی بلوک را می‌پذیرند که تمام تراکنش‌های درون آن معتبر بوده و دوباره خرج نشده باشند.
6. گره‌ها با آغاز کار روی بلوک بعدی از زنجیره و استفاده از هش بلوک پذیرفته‌شده به‌عنوان هش قبلی، پذیرش خود را اعلام می‌کنند.

گره‌ها همیشه طولانی‌ترین زنجیره را به‌عنوان زنجیره‌ی صحیح در نظر می‌گیرند و به توسعه‌ی آن ادامه می‌دهند. اگر دو گره به‌طور هم‌زمان نسخه‌های متفاوتی از بلوک بعدی را منتشر کنند، ممکن است بعضی گره‌ها یکی از آن دو را زودتر دریافت کنند. در این حالت، آن‌ها روی بلوکی که ابتدا دریافت کرده‌اند کار می‌کنند اما شاخه‌ی دیگر را نیز نگه می‌دارند تا در صورت طولانی‌تر شدن آن، به آن سوییچ کنند. اختلاف زمانی با یافتن اثبات کار بعدی و بلندتر شدن یکی از شاخه‌ها حل می‌شود، و گره‌هایی که روی شاخه‌ی کوتاه‌تر کار می‌کردند، به شاخه‌ی طولانی‌تر می‌پیوندند.

انتشار تراکنش‌های جدید لزوماً نیازی به رسیدن به همه‌ی گره‌ها ندارد. تا زمانی که به تعداد زیادی گره برسد، در نهایت وارد یکی از بلوک‌ها خواهد شد. انتشار بلوک‌ها نیز نسبت به افتادن پیام‌ها مقاوم است. اگر گره‌ای بلوکی را دریافت نکند، زمانی که بلوک بعدی را دریافت کند و متوجه شود یکی را از دست داده، آن را درخواست خواهد داد.

انگیزه

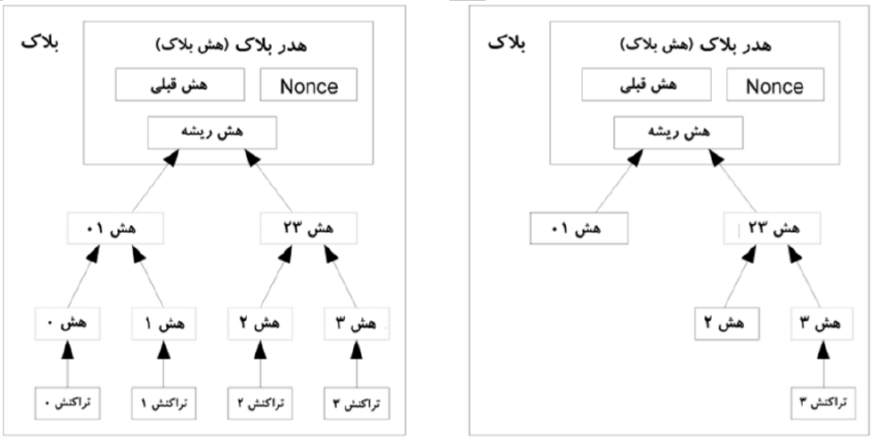
طبق قرارداد، اولین تراکنش در هر بلوک، یک تراکنش ویژه است که یک سکه‌ی جدید ایجاد کرده و آن را به خالق بلوک اختصاص می‌دهد. این سازوکار انگیزه‌ای برای گره‌ها جهت پشتیبانی از شبکه فراهم می‌کند و راهی برای توزیع اولیه‌ی سکه‌ها در گردش است، زیرا هیچ مرجع مرکزی برای انتشار سکه‌ها وجود ندارد. اضافه شدن تدریجی و ثابت سکه‌های جدید مشابه کاری است که معدن‌کاران طلا برای وارد کردن طلا به چرخه‌ی اقتصادی انجام می‌دهند؛ با این تفاوت که در اینجا زمان CPU و برق مصرف می‌شود.

انگیزه همچنین می‌تواند از کارمزد تراکنش‌ها تأمین شود. اگر مقدار خروجی یک تراکنش کمتر از مقدار ورودی آن باشد، تفاوت به‌عنوان کارمزد تراکنش به پاداش بلوک اضافه می‌شود. زمانی که تعداد مشخصی از سکه‌ها وارد گردش شد، انگیزه می‌تواند کاملاً به کارمزد تراکنش‌ها منتقل شده و عملاً بدون تورم شود.

این سیستم انگیزشی ممکن است باعث شود گره‌ها صادق بمانند. اگر مهاجم طمع‌کاری بتواند توان پردازشی بیشتری نسبت به تمام گره‌های صادق جمع‌آوری کند، باید بین استفاده از آن برای تقلب و پس‌گرفتن پرداخت‌هایش یا استفاده از آن برای تولید سکه‌های جدید انتخاب کند. منطقی‌تر است که او با رعایت قواعد سیستم عمل کند، زیرا این قواعد باعث می‌شوند نسبت به دیگران سکه‌های بیشتری به دست آورد، تا اینکه سیستم را تضعیف کرده و اعتبار دارایی خود را نیز از بین ببرد.

بازپس‌گیری فضای دیسک

زمانی که آخرین تراکنش در یک سکه تحت تعداد کافی بلوک قرار می‌گیرد، تراکنش‌های خرج‌شده قبل از آن می‌توانند برای صرفه‌جویی در فضای دیسک حذف شوند. برای تسهیل این کار بدون شکستن هش بلوک، تراکنش‌ها در یک درخت مرکل [7][2][5] هش می‌شوند، که فقط ریشه آن در هش بلوک گنجانده می‌شود. سپس بلوک‌های قدیمی می‌توانند با حذف شاخه‌های درخت فشرده شوند. هش‌های داخلی نیازی به ذخیره شدن ندارند.



تراکنش های هش شده در درخت Merkle

بعد از هرس کردن تراکنش های ۰ تا ۲ از بلاک

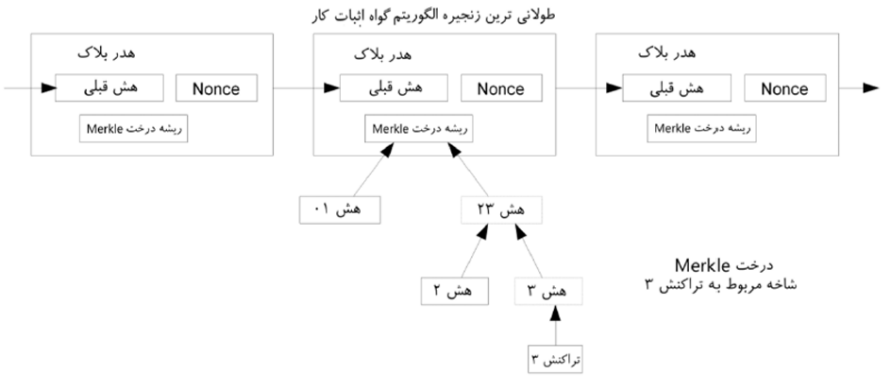
یک سرخط بلوک (Block Header) بدون تراکنش تقریباً 80 بایت حجم دارد. اگر فرض کنیم هر 10 دقیقه یک بلوک تولید شود، حاصل 80 × 6 × 24 × 365 برابر با حدود 4.2 مگابایت در سال می‌شود. با توجه به اینکه در سال 2008 کامپیوترها

معمولاً با 2 گیگابایت رم عرضه می‌شدند و طبق قانون مور، رشد سالانه‌ی حافظه حدود 1.2 گیگابایت است، حتی اگر سرخط بلوک‌ها نیاز به نگهداری در حافظه داشته باشند، فضای ذخیره‌سازی مشکلی ایجاد نخواهد کرد.

تأیید ساده پرداخت‌ها

تأیید پرداخت‌ها بدون اجرای یک نود کامل شبکه ممکن است. کاربر تنها نیاز دارد که یک نسخه از هدرهای بلاک در طولانی‌ترین زنجیره اثبات کار را نگه دارد؛ که این اطلاعات را می‌تواند با پرس‌وجو از نودهای شبکه به دست آورد تا مطمئن شود طولانی‌ترین زنجیره را دارد. سپس شاخه مرکل‌ای (Merkle Branch) که تراکنش را به بلاکی که در آن زمان‌گذاری شده متصل می‌کند، دریافت می‌کند.

کاربر نمی‌تواند تراکنش را مستقیماً بررسی کند، اما با اتصال آن به جایگاهی در زنجیره، می‌تواند ببیند که نودی از شبکه آن را پذیرفته و بلاک‌هایی که بعد از آن اضافه می‌شوند، تأییدی بر این هستند که شبکه آن را قبول کرده است.

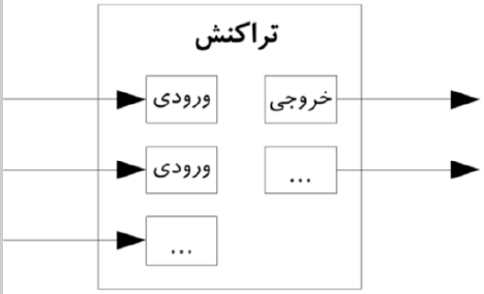


بنابراین این روش تأیید تا زمانی که نودهای صادق کنترل شبکه را در دست داشته باشند، قابل اعتماد است. اما اگر شبکه تحت سلطه مهاجم قرار گیرد، آسیب‌پذیر خواهد شد. در حالی که نودهای شبکه می‌توانند تراکنش‌ها را خودشان تأیید کنند، روش ساده‌شده ممکن است تا زمانی که مهاجم قادر به غلبه بر شبکه است، فریب بخورد. یکی از راهکارهای مقابله این است که نرم‌افزار کاربر هشدارهایی را از نودهای شبکه دریافت کند زمانی که بلاکی نامعتبر تشخیص داده می‌شود؛ سپس نرم‌افزار می‌تواند آن بلاک کامل و تراکنش‌های هشدار داده‌شده را دانلود کند تا ناسازگاری را تأیید کند.

کسب‌وکارهایی که پرداخت‌های مکرر دریافت می‌کنند، احتمالاً همچنان ترجیح می‌دهند نود کامل خود را اجرا کنند تا امنیت مستقل‌تر و تأیید سریع‌تری داشته باشند.

تجمیع و تفکیک مقدار

اگرچه امکان دارد که هر کوین را به‌صورت جداگانه مدیریت کرد، اما ساخت یک تراکنش مجزا برای هر سنت در یک انتقال، کار بسیار طاقت‌فرسا و ناکارآمدی خواهد بود. برای امکان‌پذیر شدن تجمیع و تفکیک مقدار، تراکنش‌ها شامل چندین ورودی و خروجی هستند.



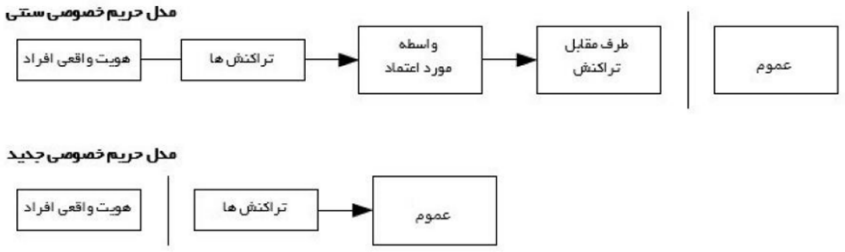
در حالت معمول، یا یک ورودی از یک تراکنش بزرگ قبلی وجود دارد یا چند ورودی برای تجمیع مقادیر کوچک‌تر. حداکثر نیز دو خروجی وجود دارد: یکی برای پرداخت و یکی برای بازگرداندن باقی‌مانده (اگر وجود داشته باشد) به فرستنده.

شایان ذکر است که انشعاب (fan-out)، یعنی جایی که یک تراکنش به چند تراکنش وابسته باشد و آن‌ها نیز به تراکنش‌های بیشتری وابسته باشند، مشکلی ایجاد نمی‌کند. هرگز نیازی به استخراج نسخه‌ی مستقل و کامل از تاریخچه‌ی یک تراکنش نیست.

حریم خصوصی

مدل سنتی بانکداری با محدود کردن دسترسی به اطلاعات، بین طرفین درگیر و یک نهاد مورد اعتماد، سطحی از حریم خصوصی را برقرار می‌کند. اما نیاز به اعلام عمومی تمام تراکنش‌ها در بیت‌کوین، این روش را غیرممکن می‌سازد.

با این حال، حریم خصوصی می‌تواند از طریق شکستن جریان اطلاعات در نقطه‌ای دیگر حفظ شود: با ناشناس نگه‌داشتن کلیدهای عمومی. عموم می‌توانند ببینند که شخصی مقدار معینی را برای شخص دیگری ارسال کرده، اما بدون هیچ اطلاعاتی که تراکنش را به فرد خاصی پیوند دهد.



این مشابه اطلاعات منتشر شده توسط بورس‌ها است که زمان و مقدار هر معامله (tape) را نشان می‌دهند، اما هویت طرفین را فاش نمی‌کنند.

به‌عنوان یک سد امنیتی اضافی، باید برای هر تراکنش از یک جفت کلید جدید استفاده شود تا مانع از اتصال تراکنش‌ها به یک مالک مشترک شود. با این وجود، برخی پیوندها هنوز اجتناب‌ناپذیر هستند، به‌ویژه در تراکنش‌های چند ورودی که لزوماً نشان می‌دهند تمام ورودی‌ها متعلق به یک مالک بوده‌اند. خطر این است که اگر هویت یک کلید افشا شود، ممکن است پیوند بین تراکنش‌ها دیگر متعلق به همان مالک نیز فاش شود.

محاسبات

در اینجا سناریویی را بررسی می‌کنیم که در آن مهاجم تلاش دارد زنجیره‌ی جایگزینی سریع‌تر از زنجیره‌ی صادق بسازد. حتی اگر در این کار موفق شود، نمی‌تواند سیستم را برای تغییرات دلخواه مانند ایجاد پول از هیچ یا تصاحب پولی که به او تعلق نداشته، باز کند.

نودها هیچ‌گاه یک تراکنش نامعتبر را به‌عنوان پرداخت نمی‌پذیرند و نودهای صادق نیز هیچ‌گاه بلاکی حاوی آن را قبول نمی‌کنند. مهاجم فقط می‌تواند تلاش کند یکی از تراکنش‌های خودش را تغییر دهد تا پولی که اخیراً خرج کرده را بازگرداند.

رقابت بین زنجیره‌ی صادق و زنجیره‌ی مهاجم را می‌توان به صورت یک «حرکت تصادفی دوجمله‌ای» مدل‌سازی کرد. رخداد موفقیت زمانی است که زنجیره‌ی صادق با یک بلاک جدید گسترش یابد (+1)، و رخداد شکست زمانی است که زنجیره‌ی مهاجم گسترش یابد.(-1)

احتمال اینکه مهاجم از یک فاصله عقب‌افتاده بتواند جبران کند، مشابه مسئله‌ی «ورشکستگی قمارباز» است. فرض کنید قماربازی با اعتبار نامحدود از یک نقطه‌ی زیان شروع کرده و تعداد نامحدودی فرصت دارد تا به سربه‌سر برسد.

ما می‌توانیم احتمال اینکه او به سربه‌سر برسد یا اینکه مهاجم به زنجیره‌ی صادق برسد را به‌صورت زیر محاسبه کنیم:

- p =احتمال اینکه نود صادق بلاک بعدی را پیدا کند
- q =احتمال اینکه مهاجم بلاک بعدی را پیدا کند
- qz =احتمال اینکه مهاجم از z بلاک عقب افتاده، برسد

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

با فرض اینکه p > q ، احتمال موفقیت مهاجم با افزایش تعداد بلاک‌هایی که عقب است، به‌صورت نمایی کاهش می‌یابد. اگر مهاجم در ابتدا خوش‌شانس نباشد، شانسش به‌سرعت به صفر میل می‌کند.

حال بررسی می‌کنیم که گیرنده یک تراکنش چقدر باید صبر کند تا مطمئن شود فرستنده نمی‌تواند تراکنش را تغییر دهد. فرض می‌کنیم فرستنده مهاجم است و می‌خواهد گیرنده را متقاعد کند که پرداخت انجام شده، اما پس از مدتی آن را برگرداند.

گیرنده بلافاصله قبل از امضای تراکنش، یک جفت کلید جدید تولید می‌کند و کلید عمومی را به فرستنده می‌دهد. این باعث می‌شود فرستنده نتواند از قبل روی یک زنجیره موازی کار کند و در لحظه‌ای مناسب آن را جایگزین کند.

پس از ارسال تراکنش، مهاجم مخفیانه روی زنجیره‌ی جایگزینی کار می‌کند. گیرنده صبر می‌کند تا تراکنش وارد یک بلاک شود و z بلاک پس از آن نیز ساخته شوند. او میزان پیشرفت مهاجم را نمی‌داند، اما فرض می‌کند نودهای صادق بلاک‌ها

را با میانگین زمان مورد انتظار ساخته‌اند؛ در این حالت پیشرفت مهاجم دارای توزیع پواسون با مقدار مورد انتظار زیر است:

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

برای به‌دست آوردن احتمال اینکه مهاجم هنوز بتواند برسد، چگالی پواسون برای هر مقدار پیشرفت ممکن او را در احتمال رسیدن از آن نقطه ضرب می‌کنیم.

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

با بازآرایی، از جمع کردن دنباله‌ی بی‌نهایت توزیع اجتناب می‌شود.

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

کد C برای محاسبه احتمال موفقیت مهاجم:

```
#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;}
```

نتایج نمونه (نمایی بودن افت احتمال z):

برای 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

برای 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

z = 30 ⟶ P = 0.0001522

z = 35 ⟶ P = 0.0000379

z = 40 ⟶ P = 0.0000095

z = 45 ⟶ P = 0.0000024

z = 50 ⟶ P = 0.0000006

برای رسیدن به احتمال کمتر از ۰.۱٪:(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 یک سیستم برای انجام تراکنش‌های الکترونیکی بدون وابستگی به اعتماد پیشنهاد کرده‌ایم. ما با چارچوب معمول سکه‌ها که از امضای دیجیتال ساخته شده‌اند آغاز کردیم، که کنترل قوی بر مالکیت فراهم می‌کند، اما بدون روشی برای جلوگیری از دوباره‌خرج کردن (Double Spending) ناقص است. برای حل این مشکل، یک شبکه همتا به همتا (P2P) با استفاده از اثبات کار (Proof-of-Work) پیشنهاد کردیم تا تاریخچه عمومی تراکنش‌ها را ثبت کند که تغییر آن برای یک مهاجم در صورتی که گره‌های صادق اکثریت قدرت پردازشی را کنترل کنند، به سرعت غیرعملی می‌شود. این شبکه در سادگی بدون ساختار خود مقاوم است. گره‌ها همزمان و با هماهنگی کم کار می‌کنند. آنها نیاز به شناسایی ندارند، زیرا پیام‌ها به مکان خاصی هدایت نمی‌شوند و تنها باید به صورت تلاش حداکثری تحویل داده شوند. گره‌ها می‌توانند به دلخواه از شبکه خارج شوند و به آن بازگردند و زنجیره اثبات کار را به عنوان مدرکی از آنچه که در زمان غیاب آنها اتفاق افتاده قبول کنند. آنها با قدرت پردازشی خود رأی می‌دهند و پذیرش بلوک‌های معتبر را با کار بر روی گسترش آنها و رد بلوک‌های نامعتبر را با امتناع از کار روی آنها نشان می‌دهند. هر قانونی و انگیزه‌ای که لازم باشد می‌تواند از طریق این مکانیزم توافقی اجرا شود.

مراجع

[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.hashcash.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.

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto

satoshin@gmx.com

www.bitcoin.org

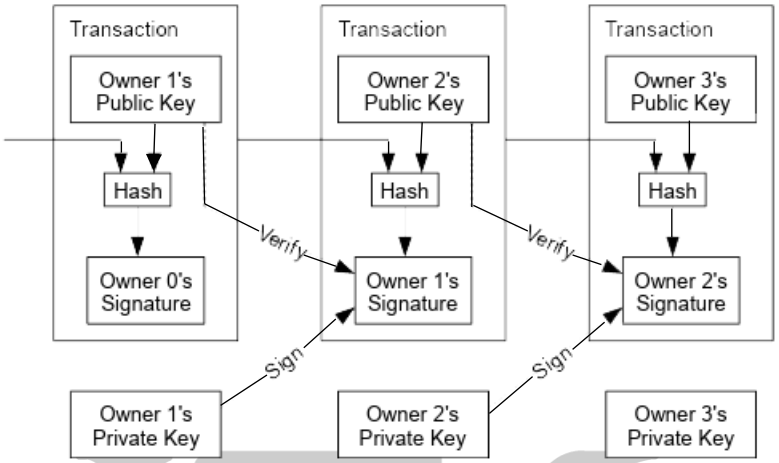
Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for nonreversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party. What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

Verify Transactions

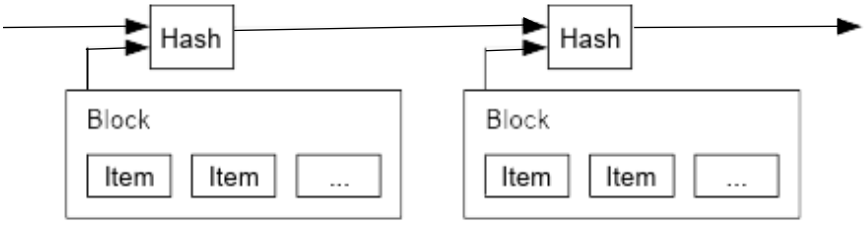
We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.



The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank. We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

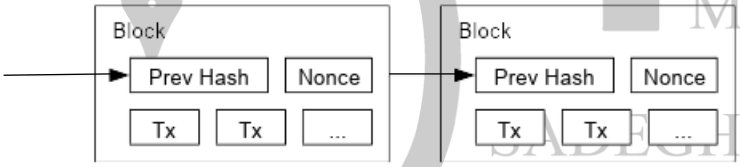
Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proofof-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash. For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added. To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving

average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.

2) Each node collects new transactions into a block.

3) Each node works on finding a difficult proof-of-work for its block.

4) When a node finds a proof-of-work, it broadcasts the block to all nodes.

5) Nodes accept the block only if all transactions in it are valid and not already spent.

6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proofof-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one. 3New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

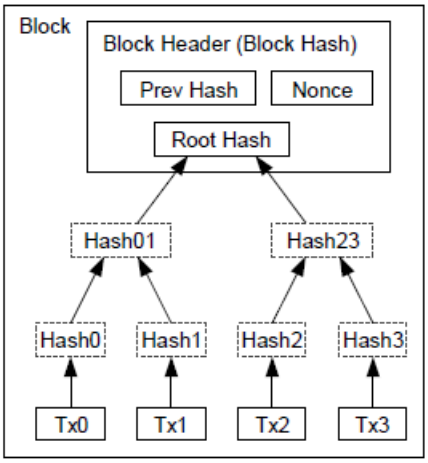
Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended. The incentive can also be funded with transaction fees. If the output value of a transaction is less than its

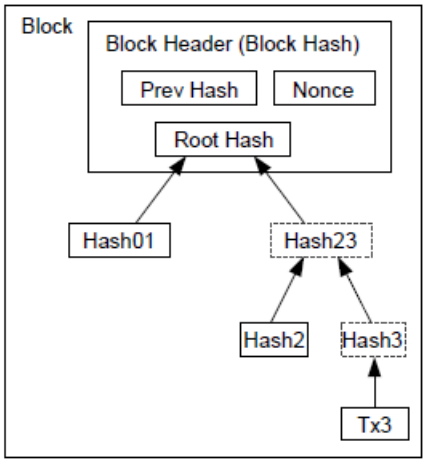
input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free. The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



Transactions Hashed in a Merkle Tree

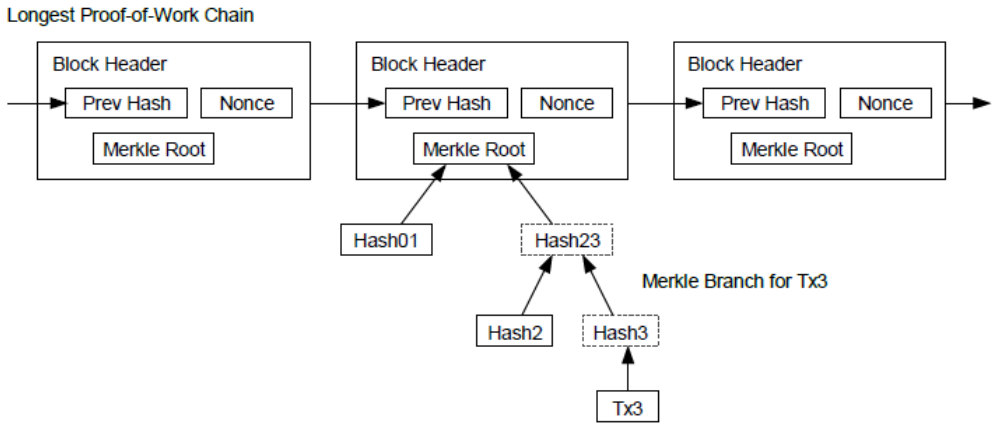


After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

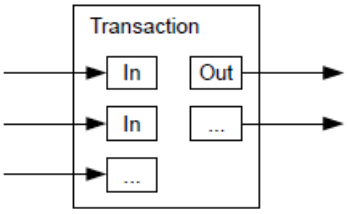


As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous

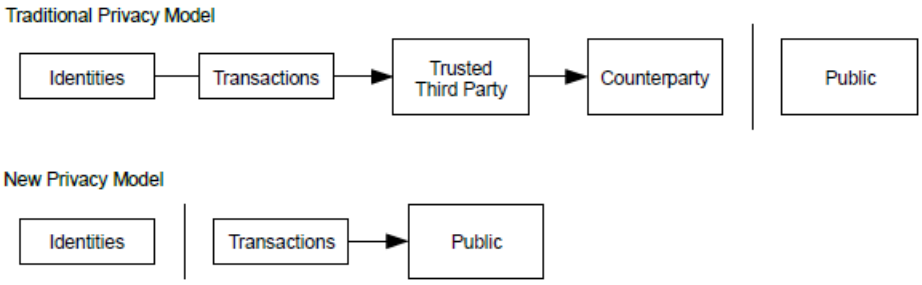
transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent. The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1. The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

- p = probability an honest node finds the next block
- q = probability the attacker finds the next block
- qz = probability the attacker will ever catch up from z blocks behind

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

Given our assumption that p > q, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind. We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late. The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment.

Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction. The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

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

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

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

Rearranging to avoid summing the infinite tail of the distribution...

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

Converting to C code...

```
#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; }
```

Running some results, we can see the probability drop off exponentially with z.

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

z=30 P=0.0001522

z=35 P=0.0000379

z=40 P=0.0000095

z=45 P=0.0000024

z=50 P=0.0000006

Solving for P less than 0.1%...

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

Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[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.hashcash.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.