Bitcoin: Um sistema de dinheiro eletrónico ponto-a-ponto

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Resumo. Uma versão puramente ponto-a-ponto de dinheiro eletrónico poderia permitir o envio de pagamentos *online* diretamente entre as partes e sem a intermediação de uma instituição financeira. As assinaturas digitais providenciam parte da solução, mas os principais benefícios perder-se-iam caso um intermediário de confiança fosse ainda necessário para prevenir o "gasto duplo". Propomos uma solução para o problema do "gasto duplo" usando uma rede ponto-por-ponto. A rede cria uma "marca temporal"[1] nas transações, convertendo-as numa corrente contínua de protocolo "prova de trabalho"[2] baseada em *hash*[3], estabelecendo um registo que não se pode modificar sem se refazer "prova de trabalho". A corrente mais longa não serve apenas como prova efetiva da sequência de eventos, mas também demonstra de que a mesma só pode ter sido originada do conjunto com maior poder de CPU. Enquanto a maioria do poder da CPU for controlada por "nós" que não cooperem entre si para atacar a própria rede, estes irão gerar a corrente mais longa e superarão os atacantes. A própria rede requer estrutura mínima. As mensagens são transmitidas na base do "melhor esforço"[4], os "nós" podem sair e regressar à rede à vontade, aceitando a corrente de "prova de trabalho" mais longa como prova do que aconteceu durante a sua ausência.

1. Introdução

O comércio na *Internet* passou a confiar quase que exclusivamente em instituições financeiras que atuam como terceiros confiáveis para processar pagamentos eletrónicos. Enquanto o sistema funciona bem o suficiente para a maioria das transações, ainda sofre com as fraquezas inerentes ao modelo baseado em confiança.

Transações totalmente irreversíveis não são realmente possíveis, já que as instituições financeiras não podem evitar ter de mediar disputas. O custo da mediação aumenta os custos de transação, limitando a tamanho mínimo prático da operação e anulando assim a possibilidade de pequenas transações casuais. E existe um custo mais amplo na perda de capacidade de fazer pagamentos irreversíveis para Serviços irreversíveis. Com a possibilidade de reversão, a necessidade de confiança cresce. Os comerciantes devem ter precaução com os seus clientes, incomodando-os para obter mais informações do que de outra forma precisariam. Uma certa percentagem de fraude é aceita como inevitável. Estes custos e incertezas de pagamento podem ser evitados pelo uso de moeda física, mas não existe mecanismo para fazer pagamentos através de um canal de comunicação sem um terceiro confiável.

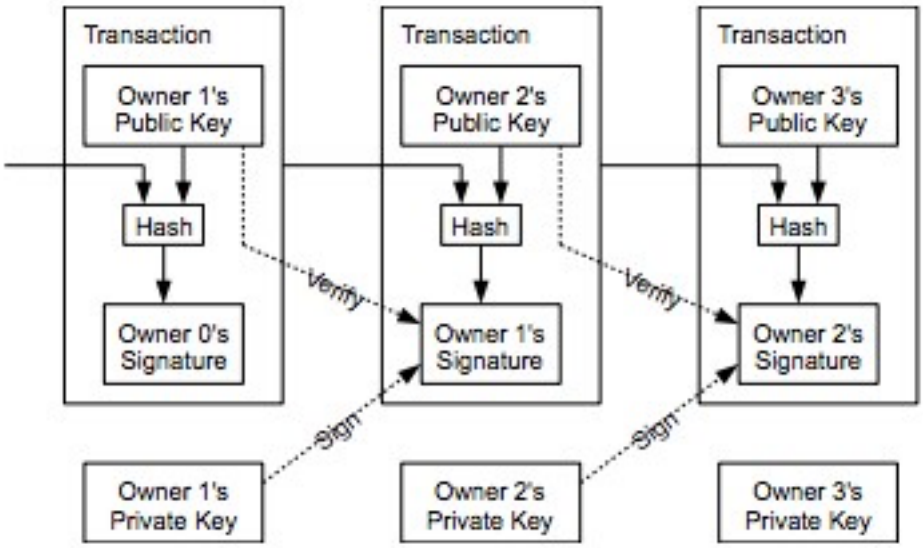
O que é preciso é um sistema de pagamento eletrónico baseado em prova criptográfica em vez de confiança, permitindo que duas partes interessadas realizem transações diretamente entre si e sem a necessidade de um terceiro. As transações que serão computacionalmente impossíveis de reverter protegerão os vendedores de fraude e

mecanismos de depósito de garantia podem ser facilmente implementados para proteger os compradores. Neste artigo propomos uma solução para o problema do "gasto duplo" usando um servidor que regista uma "marca temporal" e é distribuído ponto-a-ponto, por forma a gerar provas computacionais sobre a ordem cronológica das transações. O sistema é seguro, desde que os "nós" honestos controlem coletivamente mais potência da CPU do que qualquer grupo cooperante de "nós" atacantes.

2. Transações

Nós definimos uma moeda eletrónica como uma corrente de assinaturas digitais. Cada proprietário transfere a moeda para o próximo assinando digitalmente um *hash* da transação anterior e a chave pública do próximo proprietário e adicionando estas ao final da moeda. Um beneficiário pode verificar as assinaturas para verificar a corrente de propriedade.

O problema, obviamente, é que o beneficiário não pode verificar se um dos proprietários não gastou duas vezes a moeda. Uma solução comum é introduzir uma



autoridade central confiável, ou casa da moeda, que verifica cada transação para "gasto duplo". Após cada transação, a moeda deve ser devolvida à casa da moeda para que seja emitida nova moeda, e apenas as moedas emitidas diretamente pela casa da moeda estão livres de suspeita de "gasto duplo".

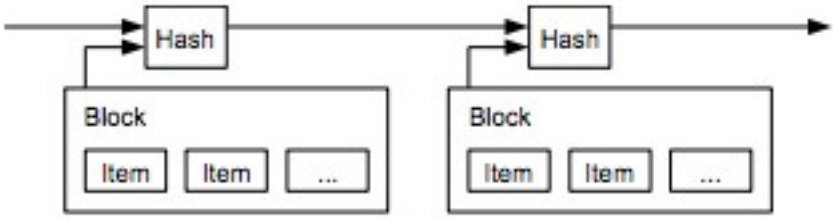
O problema com esta solução é que a sorte de todo o sistema monetário depende da empresa que controla a emissão, com cada transação ter que passar por ela, tal como um banco.

Precisamos de uma maneira para o beneficiário saber se os proprietários prévios não assinaram nenhuma nova transação. Para nossos propósitos, a transação mais antiga é a que conta, então não nos importamos sobre novas tentativas de "gasto duplo". A única maneira de confirmar a ausência de uma transação é deter o conhecimento de todas as transações. No modelo da casa da moeda, esta detém o conhecimento de todas as transações e decide qual chegou em primeiro lugar. Para atingir esse objetivo sem um terceiro confiável, as transações devem ser anunciadas publicamente [1], e precisamos de um sistema para que os participantes concordem com um único historial da ordem em que

foram recebidos. O beneficiário precisa de prova de que, no momento da transação, a maioria dos "nós" estava de acordo em que essa foi a primeira que se recebeu.

3. Servidor com "marca temporal"

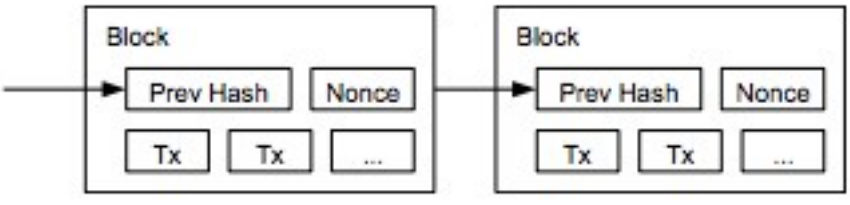
A solução que propomos começa com um servidor com registo de "marca temporal". Um servidor com registo de "marca temporal" funciona tomando uma *hash* de um bloco de itens para ser registados no tempo e publicar publicamente o *hash*, como uma publicação de jornal ou *Usenet* [2-5]. O registo temporal prova que os dados devem ter existido no tempo, obviamente, para entrar no *hash*. Cada registo temporal inclui o registo imediatamente anterior em um *hash*, formando uma corrente, com cada registo temporal adicional a reforçar os blocos anteriores.



4. O protocolo "Prova de trabalho"

Para implementar um servidor com registo de "marca temporal" distribuído numa base ponto-a-ponto, precisamos usar o protocolo "prova de trabalho" semelhante ao *Hashcash* de Adam Back [6], em vez de publicações de jornal ou *Usenet*. A "prova de trabalho" envolve procurar por um valor que, ao ser aplicada uma função de *hash* por um algoritmo como o SHA-256, o *hash* começa com um número de "zero bits". O trabalho médio requerido é exponencial no número de "zero bits" necessários e pode ser verificado executando um único *hash*.

Para a nossa rede de registo de "marca temporal", implementamos o protocolo "prova de trabalho", incrementando um *nonce*[5] no bloco até que o valor encontrado sirva para decodificar o bloco com o número de "zero bits" requeridos. Uma vez despendido esforço de CPU necessário para que se satisfaça a "prova de trabalho", o bloco não pode ser alterado sem refazer o trabalho. Como os blocos seguintes são encadeados logo após este, o trabalho para mudar o bloco inclui refazer todos os blocos posteriores.



O protocolo "prova de trabalho" também resolve o problema de determinar a representação na tomada de decisão da maioria. Se a maioria fosse determinada como "um endereço IP, um voto", esta poderia ser subvertida por qualquer pessoa capaz de alocar muitos IPs. A "prova de trabalho" é essencialmente "um CPU, um voto". A decisão

da maioria é representada pela corrente mais longa, que tem o maior esforço de "prova de trabalho" investido nela. Se a maioria do poder de processamento for controlada por "nós" honestos, a corrente honesta crescerá mais rápido e ultrapassará qualquer corrente competidora. Para modificar um bloco antigo, um atacante teria que refazer a "prova do trabalho" do bloco, de todos os blocos posteriores e depois ainda ultrapassar o atraso trabalho para com os "nós" honestos. Posteriormente mostraremos que a probabilidade de um atacante lento alcançar os honestos diminui exponencialmente, à medida em que os blocos subsequentes vão sendo adicionados à corrente honesta.

Para compensar o aumento da velocidade do *hardware* e o interesse variável em criar "nós" ativos ao longo do tempo, a dificuldade de "prova de trabalho" é determinada por uma média móvel visando um número médio de blocos por hora. Se estes são gerados muito rapidamente, a dificuldade aumenta.

5. Rede

As etapas para concretizar a rede são as seguintes:

1) As novas transações são transmitidas para todos os "nós".
2) Cada "nó" recolhe as novas transações para um bloco.
3) Cada "nó" trabalha em encontrar uma "prova de trabalho" difícil para o seu bloco.
4) Quando um "nó" encontra uma "prova de trabalho", este transmite o bloco para todos os "nós".
5) Os "nós" aceitam o bloco somente se todas as transações nele forem válidas e não houver "gasto duplo".
6) Os "nós" expressam sua aceitação do bloco trabalhando na criação do próximo bloco na corrente, usando o *hash* do bloco aceite como o *hash* anterior.

Os "nós" consideram sempre que a corrente mais longa como a correta e continuarão trabalhando estendendo-a. Se dois "nós" transmitirem diferentes versões do próximo bloco em simultâneo, alguns dos "nós" podem receber um ou o outro primeiro. Nesse caso, eles trabalham na primeira versão que receberam, mas mantêm a outra versão para o caso desta se vier a tornar a mais longa. O impasse será solucionado quando a "prova de trabalho" seguinte for encontrada e uma das versões se tornar maior. Os "nós" que então estejam a trabalhar na corrente mais curta vão mudar-se e passar a trabalhar na versão maior.

As transmissões de novas transações não necessitam necessariamente de alcançar todos os "nós". Com o alcançar da maioria dos "nós" estas, mais tarde ou mais cedo, serão incluídas num bloco. As transmissões de blocos também são tolerantes a mensagens perdidas. Se um "nó" não receber um bloco, este irá solicitá-lo quando receber o bloco seguinte e se aperceber que perdeu o anterior.

6. Incentivo

Por convenção, a primeira transação em um bloco é uma transação especial que inicia uma nova moeda de propriedade pelo criador do bloco. Isto cria um incentivo para "nós" suportarem a rede e fornece uma maneira de distribuir inicialmente moedas em circulação, uma vez que não há autoridade central para emiti-las.

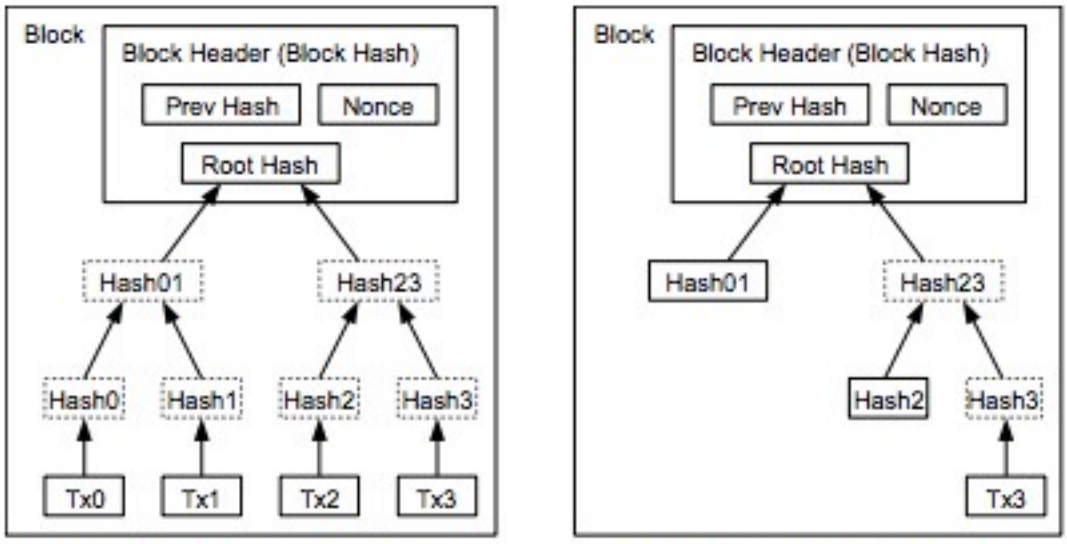
A adição constante de uma quantidade constante de moedas novas é análoga aos mineiros de ouro que estão gastando recursos para adicionar ouro à circulação. No nosso caso, trata-se tempo de processamento e eletricidade que estão a ser gastos.

O incentivo também pode ser financiado com taxas de transação. Se o valor de saída de uma transação for menos do que seu valor de entrada, a diferença é uma taxa de transação que é adicionada ao valor de incentivo de o bloco que contém a transação. Uma vez que o número pré-determinado de moedas já tenha entrado circulação, o incentivo pode passar inteiramente para as taxas de transação e com isto tornar-se completamente livre de inflação.

O incentivo pode ajudar a incentivar os "nós" a permanecerem honestos. Se um atacante ganancioso for capaz de montar mais potência de processamento do que todos os "nós" honestos, ele teria que escolher entre usá-lo para defraudar as pessoas roubando seus pagamentos, ou usá-lo para emitir novas moedas. Ele deverá considerar mais lucrativo para jogar pelas regras, já estas regras o irão favorecer com a emissão de mais moedas novas do que todos os outros, do que prejudicar o sistema e, com isso, a validade da sua própria riqueza.

7. Preocupação com o espaço de disco

Uma vez que a última transação em uma moeda é sobreposta em blocos suficientes, as transações gastas anteriormente podem ser descartadas para economizar espaço no disco. Para facilitar isso sem quebrar o *hash* do bloco, a função de *hash* é aplicada às transações em estrutura de "árvores de *Merkle*"[6] [7] [2] [5], em que apenas a raiz é incluída no *hash* do bloco. Os blocos antigos podem então ser compactados ao remover os ramos mais antigos da árvore. As *hashes* interiores não precisam de ser armazenadas.



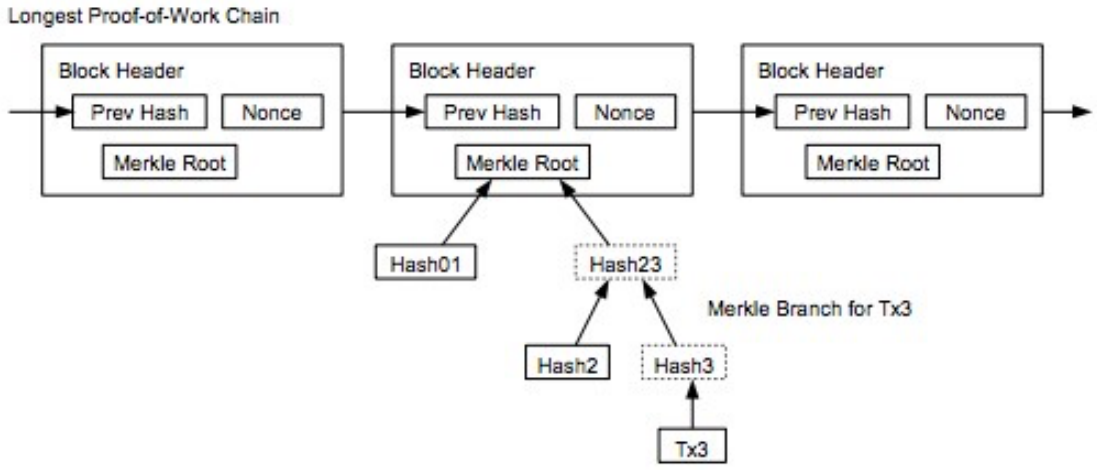
Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

Um cabeçalho de bloco sem transações deve ter cerca de 80 *bytes*. Se supusermos que os blocos são gerados a cada 10 minutos, 80 *bytes* * 6 * 24 * 365 = 4,2 MB por ano. Sendo habitual a venda de computadores com 2GB de RAM em 2008, com a "Lei de *Moore*"[7] prevendo o crescimento atual de 1,2 GB por ano, o armazenamento não deve ser um problema mesmo se os cabeçalhos dos blocos forem mantidos na memória.

8. Verificação simplificada de pagamento

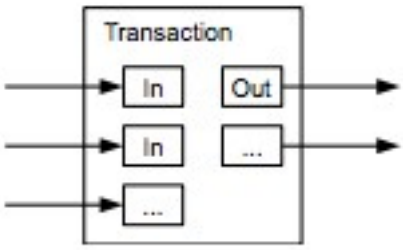
É possível verificar pagamentos sem ter de executar um "nó" de rede completo. Um utilizador só precisa manter uma cópia dos cabeçalhos de bloco da corrente de "prova de trabalho" mais longa, que ele pode obter consultando os "nós" de rede até que fique convencido que tem a corrente mais longa, e obter o ramo *Merkle* que une a transação ao bloco que está registado com a "marca temporal". Ele não pode verificar a transação por si só, mas ao ligá-la a um local da corrente, ele pode identificar o "nó" que o aceitou e os blocos adicionados após este irão confirmar que a rede a aceitou.



Como tal, a verificação é confiável enquanto os "nós" honestos controlem a rede, mas é mais vulnerável se a rede for dominada por um invasor. Enquanto os "nós" de rede podem verificar transações para si, o método simplificado pode ser enganado por transações fabricadas por um atacante, enquanto este continuar a dominar a rede. Uma estratégia para se proteger contra isso seria aceitar alertas de "nós" de rede quando estes detetem bloco um inválido, solicitando no *software* do utilizador que este descarregue bloco completo e as transações alertadas para confirmar a sua inconsistência. Negócios que recebem pagamentos frequentes vão provavelmente desejar correr os seus próprios "nós" para uma segurança mais independente e uma verificação mais rápida.

9. Combinação e dividindo valor

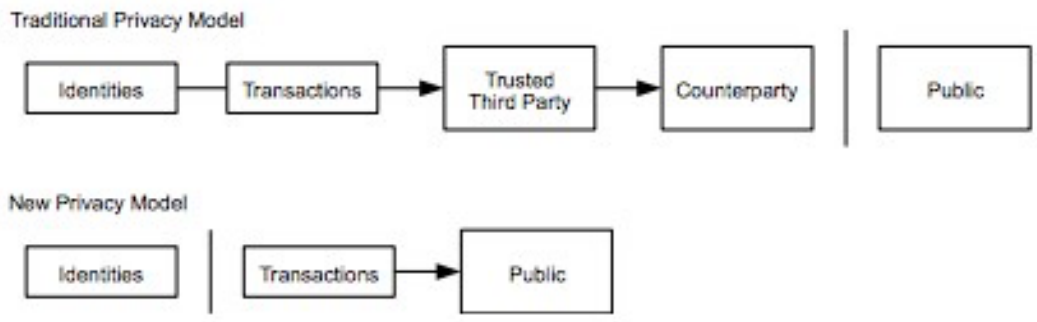
Embora seja possível lidar com moedas individualmente, seria pouco prático separar as transações para cada cêntimo a ser transferido. Para permitir que o valor seja dividido e combinado, as transações devem permitir múltiplas entradas e saídas. Normalmente haverá tanto uma única entrada de uma grande transação anterior ou múltiplas entradas combinando pequenos valores e no máximo duas saídas: uma para o pagamento, e outra para uma devolução de troco, caso exista, de volta para o emissor.



Deve-se notar que o fenómeno de *fan-out*[8], onde uma transação depende de várias transações, e essas por sua vez dependerem de muitas mais, não se supõe aqui um problema. Nunca haverá a necessidade de extrair uma cópia completa e independente do histórico de uma transação.

10. Privacidade

O modelo bancário tradicional atinge o seu nível de privacidade ao limitar o acesso à informação a apenas às partes envolvidas e a terceiros confiáveis. A necessidade de anunciar publicamente todas as transações exclui este método, mas a privacidade ainda pode ser mantida, quebrando o fluxo de informações em outro lugar: mantendo as chaves públicas anónimas. O público pode ver que alguém está enviando um valor para outra pessoa, mas sem informações que liguem a transação a qualquer individuo. Isto é semelhante ao nível de informação divulgado pelas bolsas de valores, onde o tempo e o volume dos negócios individuais, a "fita", são tornados públicos, mas sem revelar quem eram as partes.



Como elemento de segurança adicional, deverá usar-se um novo par de chaves para cada transação para evitar que sejam associados um proprietário comum. Com transações de múltipla entrada será inevitável algum tipo de vinculação, uma vez que estas revelam que a sua entrada pertence ao mesmo proprietário. O risco é que caso seja revelada a identidade do proprietário de uma chave, a sua vinculação poderia revelar outras transações que pertenceram ao mesmo proprietário.

11. Cálculos

Considerando um cenário em que um atacante tentando gerar uma corrente alternativa mais rápida do que a corrente honesta. Mesmo que isso seja conseguido, ele não vai tornar o sistema aberto para mudanças arbitrárias, como criar valor do nada ou tirar dinheiro que

nunca pertenceu ao atacante. Os "nós" são vão aceitar uma transação inválida como pagamento, e os "nós" honestos nunca aceitarão um bloco as contenha. Um atacante só pode tentar mudar uma de suas próprias transações para recuperar o dinheiro que gastou recentemente.

A corrida entre a corrente honesta e uma corrente atacante pode ser caracterizado como um "Passeio Aleatório Binomial"[9]. O evento de sucesso é a corrente honesta ser estendida por um bloco, aumentando sua liderança por +1, e o evento de falha corresponde à corrente do atacante ser estendida por um bloco, reduzindo a gap por -1. A probabilidade de um atacante se recuperar de um dado *deficit* é análoga ao problema da "ruína do jogador"[10]. Suponha que um jogador com crédito ilimitado comece com um *deficit* e jogue potencialmente um número infinito de tentativas para tentar atingir o ponto de equilíbrio. Podemos calcular a probabilidade de que ele nunca atinge o ponto de equilíbrio, ou que um atacante atinja a corrente honesta, como segue [8]:

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

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

Assumindo o pressuposto de que p > q, a probabilidade cai exponencialmente à medida que aumenta o número de blocos que o atacante tem de alcançar. Com as probabilidades contra si, se ele não tem no início um golpe inicial de sorte, suas chances vão-se desvanecendo à medida em que vai ficando para trás.

Consideremos agora quanto tempo necessita esperar o recetor de uma nova transação antes para ter a segurança de que o remetente não pode alterar a transação. Assumimos que o emissor é o atacante que quer fazer o recetor acreditar durante algum tempo que ele efetuou a transação, para então mudar a transação, devolvendo o pagamento a si mesmo algum tempo depois. O recetor receberá um aviso quando isto suceder, mas o atacante espera que já seja tarde demais.

O recetor gera um novo par de chaves e dá a chave pública ao remetente pouco antes da assinatura. Isso evita que o remetente prepare uma corrente de blocos antes do tempo trabalhando em continuamente até ter a sorte de avançar o suficiente, executando a transação em aquele momento. Uma vez que a transação é enviada, o remetente desonesto começa a trabalhar em segredo numa corrente paralela contendo uma versão alternativa da transação.

O destinatário espera até que a transação tenha sido adicionada a um bloco e que z blocos se tenham ligado posteriormente. Ele não conhece a quantidade exata de progresso que o atacante fez, mas assumindo que os blocos honestos levassem o tempo esperado médio por bloco, o potencial do atacante o progresso será uma distribuição de Poisson[11] com valor esperado:

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

Para obter a probabilidade de que o atacante ainda possa recuperar o atraso, multiplicamos a densidade de *Poisson* para cada um dos progressos que ele poderia ter realizado com a probabilidade de conseguir alcançar desde esse ponto:

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

Reorganizando para evitar a soma da cauda infinita da distribuição ...

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

Convertendo para o código 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;
}
```

Executando alguns resultados, podemos ver a probabilidade cair exponencialmente à medida que aumenta 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
```

Resolvendo para P menos de 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
```

12. Conclusão

Propusemos um sistema para transações eletrónicas sem se depender da confiança. Começamos com a estrutura comum de moedas feitas a partir de assinaturas digitais, que providenciam forte controle de propriedade, mas está incompleta sem uma maneira de evitar o "gasto duplo". Para resolver isso, nós propusemos uma rede ponto-por-ponto usando "prova de trabalho" para registar um histórico público de transações que rapidamente se torna computacionalmente impraticável para um atacante modificar se os "nós" honestos controlem a maioria do poder computacional. A rede é robusta dada a sua simplicidade não estruturada. Os "nós" trabalham todos ao mesmo tempo e com pouca coordenação entre si. Eles não precisam ser identificados, uma vez que as mensagens não são encaminhadas para nenhum local em particular e apenas precisam de ser entregues na base da "melhor forma possível". Os "nós" podem deixar e regressar à rede livremente,

aceitando a corrente de “prova de trabalho” como prova de que aconteceu na sua ausência. Eles votam com o poder de processamento computacional, expressando sua aceitação de blocos válidos ao trabalhar em estendê-los e rejeitando os blocos inválidos ao recusar-se a trabalhar em eles. Quaisquer regras e incentivos necessários podem ser aplicados com este mecanismo de consenso.

Referências

[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.

Nota do tradutor. Para esta tradução foi respeitada a estética do documento original e seus esquemas, assim como a tipografia e estrutura.

Criado a 18.fev 2018 por @anthagas para Bitcoin Portugal

[1] https://pt.wikipediarg/wiki/Marca_temporal
[2] https://pt.wikipedia.org/wiki/Prova_de_trabalho
[3] https://pt.wikipedia.org/wiki/Função_hash
[4] https://pt.wikipedia.org/wiki/Best_effort
[5] https://pt.wikipedia.org/wiki/Nonce
[6] https://pt.wikipedia.org/wiki/Árvores_de_Merkle
[7] https://pt.wikipedia.org/wiki/Lei_de_Moore
[8] https://pt.wikipedia.org/wiki/Fan-out
[9] https://pt.wikipedia.org/wiki/Passeio_aleatório
[10] https://en.wikipedia.org/wiki/Gambler’s_ruin
[11] https://pt.wikipedia.org/wiki/Distribuição_de_Poisson