Bitcoin: Um Sistema Eletrônico de Dinheiro Ponto-a-Ponto[1]

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

Traduzido para o Português (PT-BR) por
criptoficina.com

Resumo. Uma versão puramente ponto-a-ponto (peer-to-peer) de dinheiro eletrônico que permitiria que pagamentos on-line fossem enviados diretamente de uma parte para outra sem passar por uma instituição financeira. As assinaturas digitais fornecem parte da solução, mas os principais benefícios são perdidos se ainda for necessário um terceiro de confiança para evitar gastos duplos (*double-spending*[2]). Propomos uma solução para o problema de gastos duplos usando uma rede ponto-a-ponto. A rede imprime as transações, colocando-as em uma cadeia contínua de prova de trabalho (*proof-of-work*[3]) baseada em *hash*[4], formando um registro que não pode ser alterado sem refazer a prova de trabalho. A cadeia mais longa não serve apenas como prova da seqüência de eventos testemunhada, mas prova de que ela veio do *pool* de CPU mais potente. Enquanto a maioria da potência do CPU for controlada por nós que não cooperam para atacar a rede, eles gerarão a cadeia mais longa para criar vantagem sobre os invasores. A própria rede requer estrutura mínima. As mensagens são transmitidas de acordo com o melhor esforço (*best-effort*[5]), e os nós podem sair e se juntarem à rede à vontade, aceitando a cadeia de prova de trabalho mais longa como prova do que aconteceu enquanto estavam ausentes.

1. Introdução

O comércio na Internet passou a confiar quase exclusivamente em instituições financeiras que atuam como terceiros de confiança para processar pagamentos eletrônicos. Embora o sistema funcione bem o suficiente para a maioria das transações, ele 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 a mediante disputas. O custo da mediação aumenta os custos de transação, limitando o tamanho mínimo da transação prática e cortando a possibilidade de pequenas transações casuais, e há um custo mais amplo na perda de capacidade de fazer pagamentos não-reversíveis para serviços não-reversíveis. Com a possibilidade de reversão, a necessidade de confiança se espalha. Os comerciantes devem se preocupar com seus clientes, buscando obter mais informações do que de outra forma precisariam. Uma certa porcentagem de fraude é aceita como inevitável. Esses custos e incertezas de pagamento podem ser evitados pessoalmente usando a moeda física, mas não há mecanismo para fazer pagamentos através de um canal de comunicação sem uma parte confiável.

É necessário um sistema de pagamento eletrônico baseado em prova criptográfica em vez de

[1] *Peer-to-peer* https://pt.wikipedia.org/wiki/Peer-to-peer
[2] *Double-spending* https://pt.wikipedia.org/wiki/Gasto_Duplo
[3] *Proof-of-work* https://pt.wikipedia.org/wiki/Prova_de_trabalho
[4] *Hash* https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_hash
[5] *Best-effort* https://en.wikipedia.org/wiki/Best-effort_delivery

confiança, permitindo que duas partes interessadas façam transações diretamente entre elas sem a necessidade de um terceiro de confiança. As transações computacionalmente impossíveis de reverter protegerão os vendedores contra à fraude, e os mecanismos de custódia de garantia podem ser facilmente implementados para proteger os compradores. Neste artigo, propomos uma solução para o problema de gastos duplos usando um servidor de *timestamp* distribuído ponto-a-ponto para gerar comprovação computacional da ordem cronológica das transações. O sistema é seguro, desde que os nós (*nodes*[6]) controlem genuinamente e coletivamente mais energia da CPU do que qualquer outro grupo de nós invasores.

2. Transações

Nós definimos uma moeda eletrônica como uma cadeia de assinaturas digitais. Cada proprietário transfere a moeda para a próxima 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 cadeia de propriedade.

O problema, é claro, é que o beneficiário não pode verificar se um dos proprietários não gastou a mesma moeda duas vezes. Uma solução comum é introduzir uma autoridade central confiável, ou casa da moeda, que verifica todas as transações para gastos duplos. Após cada transação, a moeda deve ser devolvida à casa da moeda para emitir uma nova moeda, e apenas as moedas emitidas diretamente da casa da moeda são confiáveis para que não sejam gastas duas vezes. O problema com esta solução é que o destino de todo o sistema monetário depende da empresa que administra a casa da moeda, pela qual passa cada transação, assim como um banco.

Precisamos de um caminho para o beneficiário saber que os proprietários anteriores não assinaram quaisquer transações anteriores. Para nossos propósitos, a transação mais antiga é aquela que conta, portanto, não nos preocupamos com as tentativas posteriores de gastar duas vezes. A única maneira de confirmar a ausência de uma transação é estar ciente de todas as transações. No modelo baseado na casa da moeda, ela estava ciente de todas as transações e decide quais chegaram primeiro. Para realizar isso sem uma parte confiável, as transações devem ser publicamente anunciadas [1], e precisamos de um sistema para que os participantes concordem com um único histórico da ordem em que foram recebidos. O beneficiário precisa da prova de que, no momento de cada transação, a maioria dos nós estavam de acordo que aquela foi a primeira recebida.

3. Servidor *Timestamp*

A solução que propomos começa com um servidor *timestamp*. Um servidor *timestamp* funciona tomando um *hash* de um bloco de itens para ser confirmado e divulgado amplamente, como em um jornal ou publicação Usenet [2-5]. O *timestamp* prova que os dados devem ter existido no momento, obviamente, para entrar no *hash*. Cada confirmação da data/hora inclui a confirmação da data/hora anterior em seu *hash*, formando uma cadeia, com cada *timestamp* adicional que reforça os antes dele.

4. Prova de Trabalho (*Proof-of-Work*[3])

Para implementar um servidor de *timestamp* distribuído em uma base ponto-a-ponto, precisamos usar um sistema de prova de trabalho semelhante ao *Hashcash* de Adam Back [6], em vez de jornal ou *posts* Usenet. A prova de trabalho envolve a verificação de um valor que, quando *hasheado*, como o SHA-256, o *hash* começa com uma série de zero bits. O trabalho médio requerido é exponencial ao número de zero bits necessários e pode ser verificado executando um único *hash*.

Para a nossa rede de timestamp, implementamos a prova de trabalho incrementando um *nonce*[7] no bloco até encontrar um valor que dê ao hash do bloco os bits zero necessários. Uma vez que o

[6] *Nós/Nodes* https://criptoficina.com/glossario-de-termos-criptomoedas/
[7] *Nonce* https://en.wikipedia.org/wiki/Cryptographic_nonce

esforço da CPU foi gasto para satisfazer a prova de trabalho, o bloco não pode ser alterado sem refazer o trabalho. À medida que os blocos posteriores são acorrentados após ele, o trabalho para mudar o bloco incluiria refazer todos os blocos depois dele.

A prova de trabalho também resolve o problema de determinar a representação na tomada de decisão maioritária. Se a maioria fosse baseada em um endereço IP - um voto, poderia ser subvertida por qualquer pessoa capaz de alocar muitos IPs. A prova de trabalho é essencialmente uma CPU-um-voto. A decisão maioritária é representada pela cadeia mais longa, que tem o maior esforço de prova de trabalho investido nele. Se a maioria do poder da CPU for controlada por nós (*nodes*) genuínos, a cadeia genuína crescerá mais rápido e superará todas as cadeias concorrentes. Para modificar um bloco passado, um atacante teria que refazer a prova do trabalho do bloco e todos os blocos depois dele e depois recuperar o atraso no trabalho dos nós genuínos. Mostramos mais tarde que a probabilidade de um retardador mais lento se recuperar diminui exponencialmente à medida que os blocos subseqüentes são adicionados.

Para compensar o aumento da velocidade do hardware e o interesse variável em executar os nós ao longo do tempo, a dificuldade de prova de trabalho é determinada por uma média móvel direcionada a um número médio de blocos por hora. Se eles são gerados muito rápido, a dificuldade aumenta.

5. Rede

As etapas para executar a rede são as seguintes:

1) Novas transações são transmitidas para todos os nós.
2) Cada nó recolhe novas transações para um bloco.
3) Quando um nó encontra uma prova de trabalho, ele transmite o bloco para todos os nós.
4) Os nós aceitam o bloco somente se todas as transações nele forem válidas e já não estiverem gastas.
5) Os nós expressam sua aceitação do bloco trabalhando na criação do próximo bloco da cadeia, usando o hash do bloco aceito como o hash anterior.

Os nós sempre consideram que a cadeia mais longa é a correta e continuará trabalhando para estendê-la. Se dois nós transmitirem diferentes versões do próximo bloco simultaneamente, alguns nós podem receber um ou outro primeiro. Nesse caso, eles trabalham no primeiro que receberam, mas salve o outro ramo no caso de se tornar mais longo. O empate será resolvido quando a próxima prova de trabalho for encontrada e um ramo se tornar mais longo; os nós que estavam trabalhando no outro ramo mudarão para o mais longo.

Novas transmissões de transações não necessitam necessariamente alcançar todos os nós. Enquanto eles chegarem a muitos nós, eles entrarão em um bloco antes disso. As transmissões de blocos também são tolerantes às mensagens descartadas. Se um nó não receber um bloco, ele o solicitará quando receber o próximo bloco e perceber que faltou um.

6. Incentivo

Por convenção, a primeira transação em um bloco é uma transação especial que inicia uma nova moeda de propriedade do criador do bloco. Isso acrescenta um incentivo para os nós (*nodes*) suportarem a rede e fornecerem 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 mineradores de ouro gastando recursos para adicionar ouro à circulação. No nosso caso, é tempo de CPU e eletricidade que é gasto.

O incentivo também pode ser financiado com taxas de transação. Se o valor de saída de uma transação for inferior ao valor de entrada, a diferença é uma taxa de transação que é adicionada ao valor de incentivo do bloco que contém a transação. Uma vez que um número predeterminado de moedas entrou em circulação, o incentivo pode transitar inteiramente para taxas de transação e ser

completamente livre de inflação.

O incentivo pode ajudar a encorajar os nós a permanecerem genuínos. Se um atacante ganancioso for capaz de montar mais poder de CPU do que todos os nós genuínos, ele teria que escolher entre usá-lo para defraudar as pessoas roubando seus pagamentos ou usando-o para gerar novas moedas. Ele deveria achar mais lucrativo jogar com as regras, regras que o favoreçam com mais moedas novas do que todos os outros combinados, do que prejudicar o sistema e a validade de sua própria riqueza.

7. Recuperando Espaço em Disco

Uma vez que a transação mais recente em uma moeda está enterrada em blocos suficientes, as transações gastas antes podem ser descartadas para economizar espaço no disco. Para facilitar isso sem quebrar o *hash* do bloco, as transações são *hasheadas* em Merkle Tree [7] [2] [5], com apenas a raiz incluída no *hash* do bloco. Os blocos antigos podem então ser compactados espreitando ramos da árvore. Os *hashes* interiores não precisam ser armazenados.

Um cabeçalho de bloco (*block header*) sem transações seria de cerca de 80 bytes. Supomos que os blocos são gerados a cada 10 minutos, 80 bytes * 6 * 24 * 365 = 4,2 MB por ano. Com os sistemas de computador que normalmente são vendidos com 2 GB de RAM em 2008, e a Lei de Moore, que prevê um crescimento atual de 1,2 GB por ano, o armazenamento não deve ser um problema, mesmo que os cabeçalhos dos blocos forem mantidos na memória.

8. Verificação de Pagamento Simplificado

É possível verificar pagamentos sem executar um nó de rede completo. Um usuário só precisa manter uma cópia dos cabeçalhos de bloco da cadeia de prova de trabalho mais longa, que ele pode obter consultando nós de rede até que ele esteja convencido de que ele tem a cadeia mais longa e obter o ramo Merkle que liga a transação ao bloco que está com *timestamped*. Ele não pode verificar a transação por si mesmo, mas, ligando-a para um lugar na cadeia, ele pode ver que um nó de rede aceitou, e os blocos foram adicionados depois de confirmar que a rede aceitou.

Como tal, a verificação é confiável enquanto os nós genuinos controlem a rede, mas é mais vulnerável se a rede for dominada por um invasor. Enquanto os nós de rede podem verificar as transações por elas mesmas, o método simplificado pode ser enganado pelas transações fabricadas por um atacante enquanto o atacante continuar a superar a rede. Uma estratégia para se proteger contra isso seria aceitar alertas de nós de rede quando eles detectarem um bloco inválido, solicitando que o software do usuário baixe o bloco completo e as transações alertadas para confirmar a inconsistência. As empresas que recebem pagamentos frequentes provavelmente ainda querem executar seus próprios nós para obter uma segurança mais independente e uma verificação mais rápida.

9. Combinando e Dividindo o Valor

Embora seja possível lidar com moedas individualmente, não seria difícil fazer uma transação separada por cada centavo em uma transferência. Para permitir que o valor seja dividido e combinado, as transações contêm várias entradas e saídas. Normalmente, haverá uma única entrada de uma transação anterior maior ou múltiplas entradas que combinem quantidades menores e, no máximo, duas saídas: uma para o pagamento e uma que retorna a mudança, se houver, de volta ao remetente.

Deve-se notar que *fan-out*[8], onde uma transação depende de várias transações, e essas transações

[8] *Fan-out* https://en.wikipedia.org/wiki/Fan-out

dependem de muitas outras mais, não é um problema aqui. Nunca há a necessidade de extrair uma cópia autônoma completa do histórico de uma transação.

10. Privacidade

O modelo bancário tradicional atinge um nível de privacidade, limitando o acesso à informação às partes envolvidas e ao terceiro de confiança. A necessidade de anunciar todas as transações impossibilita publicamente esse método, mas a privacidade ainda pode ser mantida através da quebra do fluxo de informações em outro lugar: mantendo as chaves públicas anônimas. O público pode ver que alguém está enviando um valor para outra pessoa, mas sem informações que ligam a transação a ninguém. Isso é semelhante ao nível de informação divulgado pelas bolsas de valores, onde o tempo e o tamanho dos negócios individuais, o "*tape*", são tornados públicos, mas sem contar quem eram as partes.

Como um firewall adicional, um novo par de chaves deve ser usado para cada transação para mantê-los vinculados a um proprietário comum. Alguns links ainda são inevitáveis com transações de entrada múltipla, o que necessariamente revela que suas entradas foram de propriedade de um mesmo proprietário. O risco é que se o proprietário de uma chave for revelado, a ligação pode revelar outras transações que pertencem ao mesmo proprietário.

11. Cálculos

Consideramos o cenário de um atacante tentando gerar uma cadeia alternativa mais rápida do que a cadeia genuína. Mesmo que isso seja realizado, ele não abre o sistema para mudanças arbitrárias, como criar valor fora do ar ou tirar dinheiro que nunca pertenceu ao atacante. Os nós não aceitarão uma transação inválida como pagamento, e os nós genuínos nunca aceitarão um bloco que os contenha. Um invasor só pode tentar mudar uma das suas próprias transações para recuperar o dinheiro que gastou recentemente.

A corrida entre a cadeia genuína e uma cadeia atacante pode ser caracterizada como uma caminhada binomial aleatória. O evento de sucesso é a cadeia genuína sendo estendida por um bloco, aumentando a liderança por +1, e o evento de falha é a cadeia do atacante sendo estendida por um bloco, reduzindo a diferença em -1.

A probabilidade de um atacante se recuperar de um determinado déficit é análoga ao problema da ruína do jogador. Suponha que um jogador com crédito ilimitado comece com um déficit e jogue potencialmente um número infinito de tentativas para tentar atingir o ponto de equilíbrio. Podemos calcular a probabilidade de que ele alcance o ponto de equilíbrio, ou que um atacante atinja a cadeia genuína, como segue [8]:

p = probabilidade de um nó genuíno encontrar o próximo bloco
q = probabilidade do atacante encontrar o próximo bloco
q_z = probabilidade do atacante se recuperar dos blocos z de trás

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

Dado o nosso pressuposto de que $p > q$, a probabilidade diminui exponencialmente à medida que o número de blocos que o atacante tem para alcançar aumenta. Com as chances contra ele, se ele não faz uma jogada de sorte antecipada, suas chances tornam-se cada vez mais pequenas ficando muito para trás.

Agora consideramos quanto tempo o destinatário de uma nova transação precisa esperar antes de estar suficientemente seguro de que o remetente não pode alterar a transação. Nós assumimos que o remetente é um atacante que quer fazer o destinatário acreditar que ele o pagou por um tempo,

e depois alterar para pagar a si mesmo depois de algum tempo. O destinatário será alertado quando isso acontecer, mas o remetente espera que seja muito tarde.

O receptor gera um novo par de chaves e dá a chave pública ao remetente pouco antes de assinar. Isso evita que o remetente prepare uma cadeia de blocos antes do tempo, trabalhando continuamente até que ele tenha a sorte de avançar o suficiente, então executando a transação naquele momento. Uma vez que a transação é enviada, o remetente desonesto começa a trabalhar em segredo em uma cadeia paralela contendo uma versão alternativa de sua transação.

O destinatário espera até que a transação tenha sido adicionada a um bloco e os blocos z foram vinculados após ele. Ele não conhece a quantidade exata de progresso que o atacante fez, mas assumindo que os blocos genuínos levaram o tempo esperado médio por bloco, o progresso potencial do atacante será uma distribuição de *Poisson*[9] com um valor esperado:

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

Para obter a probabilidade do atacante ainda poder alcançar agora, multiplicamos a densidade de Poisson por cada quantidade de progresso que ele poderia ter feito com a probabilidade de conseguir alcançar esse ponto:

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

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 o código para a linguagem 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 com z.

[9] *Poisson* https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_de_Poisson

```
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 menor que 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

Propomos um sistema para transações eletrônicas sem depender na confiança. Começamos com o quadro usual de moedas feitas a partir de assinaturas digitais, que proporciona um forte controle da propriedade, mas está incompleta sem uma maneira de evitar gastos duplos. Para resolver isso, propusemos uma rede ponto-a-ponto usando a prova de trabalho para registrar um histórico público de transações que rapidamente se torna computacionalmente impraticável para que um invasor mude se os nós (*nodes*) genuínos que controlam a maioria da potência da CPU. A rede é robusta em sua simplicidade não estruturada. Os nós funcionam de uma só vez com pouca coordenação. Eles não precisam ser identificados, uma vez que as mensagens não são encaminhadas para nenhum local específico e só precisam ser entregues de acordo com o melhor esforço. Os nós podem sair e se juntarem à rede à vontade, aceitando a cadeia de prova de trabalho como prova do que aconteceu enquanto eles tinham desaparecido. Eles votam com o poder da CPU, expressando sua aceitação de blocos válidos trabalhando em estendê-los e rejeitando blocos inválidos, recusando-se a trabalhar neles. 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 da tradução:

Esta tradução respeitou a formatação do documento original, assim como os tamanhos de fontes, títulos, etc. Algumas expressões se mantiveram na língua original, outras foram traduzidas, porém empregadas suas respectivas correspondencias na língua original (inglês), ou nas notas de rodapé ou ao lado dos termos entre parênteses.

As notas de rodapé foram adicionadas ao texto original para facilitar a compreensão dos termos e fornecer mais informações ao leitor.