Bitcoin: Um Sistema de Dinheiro Eletrônico Ponto a Ponto

Satoshi Nakamoto

satoshin@gmx.com

www.bitcoin.org

Tradução para o português do Brasil

Resumo. Uma versão puramente ponto a ponto de dinheiro eletrônico permitiria que pagamentos online fossem enviados diretamente de uma parte para outra sem passar por uma instituição financeira. Assinaturas digitais fornecem parte da solução, mas os principais benefícios se perdem se ainda for necessário um terceiro confiável para impedir o gasto duplo. Propomos uma solução para o problema do gasto duplo usando uma rede ponto a ponto. A rede registra o tempo das transações ao agrupá-las por hash em uma cadeia contínua de prova de trabalho baseada em hashes, formando um registro que não pode ser alterado sem refazer a prova de trabalho. A cadeia mais longa serve não apenas como prova da sequência de eventos observada, mas também como prova de que ela veio do maior conjunto de poder computacional de CPU. Enquanto a maioria do poder computacional de CPU for controlada por nós que não estejam cooperando para atacar a rede, eles gerarão a cadeia mais longa e superarão os atacantes. A própria rede exige uma estrutura mínima. As mensagens são transmitidas com base no melhor esforço, e os nós podem sair e retornar à rede à vontade, aceitando a cadeia de prova de trabalho mais longa como prova do que aconteceu enquanto estiveram ausentes.

1. Introdução

O comércio na Internet passou a depender quase exclusivamente de instituições financeiras que atuam como terceiros confiáveis para processar pagamentos eletrônicos. Embora o sistema funcione suficientemente bem para a maioria das transações, ele ainda sofre com as fraquezas inerentes ao modelo baseado em confiança. Transações completamente irreversíveis não são realmente possíveis, pois as instituições financeiras não conseguem evitar a mediação de disputas. O custo da mediação aumenta os custos de transação, limita o tamanho mínimo prático das transações e elimina a possibilidade de pequenas transações casuais. Há também um custo mais amplo na perda da capacidade de realizar pagamentos irreversíveis por serviços irreversíveis. Com a possibilidade de reversão, a necessidade de confiança se espalha. Comerciantes precisam desconfiar de seus clientes, exigindo deles mais informações do que de outra forma seria necessário. Uma certa porcentagem de fraude é aceita como inevitável. Esses custos e incertezas de pagamento podem ser evitados presencialmente com o uso de dinheiro físico, mas não existe nenhum mecanismo para realizar pagamentos por um canal de comunicação sem uma parte confiável.

O que se faz necessário é um sistema de pagamento eletrônico baseado em prova criptográfica em vez de confiança, permitindo que duas partes interessadas transacionem diretamente entre si sem a necessidade de um terceiro confiável. Transações computacionalmente impraticáveis de reverter protegeriam os vendedores contra fraudes, e mecanismos rotineiros de custódia poderiam ser facilmente implementados para proteger os compradores. Neste artigo, propomos uma solução para o problema do gasto duplo usando um servidor distribuído de carimbo de tempo ponto a ponto para gerar uma prova computacional da ordem cronológica das transações. O sistema é seguro enquanto os nós honestos controlarem coletivamente mais poder computacional de CPU do que qualquer grupo cooperado de nós atacantes.

2. Transações

Definimos uma moeda eletrônica como uma cadeia de assinaturas digitais. Cada proprietário transfere a moeda para o próximo ao assinar digitalmente um hash da transação anterior e a chave pública do

próximo proprietário, acrescentando esses dados ao final da moeda. Um recebedor pode verificar as assinaturas para confirmar a cadeia de propriedade.

Figura 1. No diagrama original, cada transação contém a chave pública do novo proprietário, um hash e a assinatura do proprietário anterior. As setas indicam a assinatura feita com a chave privada e a verificação com a chave pública.

O problema, naturalmente, é que o recebedor não consegue verificar se um dos proprietários não gastou a moeda duas vezes. Uma solução comum é introduzir uma autoridade central confiável, ou casa emissora, que verifica todas as transações contra gasto duplo. Após cada transação, a moeda precisa retornar à casa emissora para a emissão de uma nova moeda, e somente moedas emitidas diretamente por ela são consideradas confiáveis contra gasto duplo. O problema dessa solução é que o destino de todo o sistema monetário depende da empresa que opera a casa emissora, com todas as transações tendo de passar por ela, assim como em um banco.

Precisamos de uma forma para que o recebedor saiba que os proprietários anteriores não assinaram transações anteriores. Para nossos objetivos, a transação mais antiga é a que vale; portanto, não nos importamos com tentativas posteriores de gasto duplo. A única maneira de confirmar a ausência de uma transação é ter conhecimento de todas as transações. No modelo baseado em uma casa emissora, ela tinha conhecimento de todas as transações e decidia qual havia chegado primeiro. Para realizar isso sem uma parte confiável, as transações devem ser anunciadas publicamente [1], e precisamos de um sistema para que os participantes concordem com uma única história sobre a ordem em que elas foram recebidas. O recebedor precisa de prova de que, no momento de cada transação, a maioria dos nós concordou que ela foi a primeira recebida.

3. Servidor de Carimbo de Tempo

A solução que propomos começa com um servidor de carimbo de tempo. Um servidor de carimbo de tempo funciona tomando o hash de um bloco de itens a serem carimbados no tempo e publicando amplamente esse hash, por exemplo, em um jornal ou em uma postagem na Usenet [2-5]. O carimbo de tempo prova que os dados deveriam existir naquele momento, obviamente, para que pudessem entrar no hash. Cada carimbo de tempo inclui o carimbo de tempo anterior em seu hash, formando uma cadeia, com cada carimbo adicional reforçando os anteriores.

Figura 2. O diagrama original mostra blocos encadeados: cada bloco contém itens, gera um hash e inclui o hash anterior, criando uma sequência de carimbos de tempo.

4. Prova de Trabalho

Para implementar um servidor distribuído de carimbo de tempo em uma base ponto a ponto, precisaremos usar um sistema de prova de trabalho semelhante ao Hashcash de Adam Back [6], em vez de publicações em jornais ou na Usenet. A prova de trabalho envolve a busca por um valor que, quando submetido a hash, como pelo SHA-256, faça com que o hash comece com certo número de bits zero. O trabalho médio necessário cresce exponencialmente conforme aumenta o número de bits zero exigidos, e pode ser verificado executando-se um único hash.

Para nossa rede de carimbo de tempo, implementamos a prova de trabalho incrementando um nonce no bloco até que seja encontrado um valor que faça o hash do bloco apresentar os bits zero exigidos. Depois que o esforço de CPU foi gasto para fazê-lo satisfazer a prova de trabalho, o bloco não pode ser alterado sem refazer esse trabalho. Como blocos posteriores são encadeados depois dele, o trabalho para alterar o bloco incluiria refazer todos os blocos posteriores.

Figura 3. O diagrama original mostra blocos com hash anterior, nonce e transações, unidos por hashes sucessivos.

A prova de trabalho também resolve o problema de determinar a representação em decisões por maioria. Se a maioria fosse baseada em um endereço IP, um voto, ela 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 cadeia mais longa, que contém o maior esforço de prova de

trabalho investido nela. Se a maioria do poder computacional de CPU for controlada por nós honestos, a cadeia honesta crescerá mais rapidamente e superará quaisquer cadeias concorrentes. Para modificar um bloco passado, um atacante teria de refazer a prova de trabalho desse bloco e de todos os blocos posteriores, e então alcançar e superar o trabalho dos nós honestos. Mostraremos adiante que a probabilidade de um atacante mais lento alcançar a cadeia diminui exponencialmente conforme blocos subsequentes são adicionados.

Para compensar o aumento da velocidade do hardware e a variação do interesse em executar nós ao longo do tempo, a dificuldade da prova de trabalho é determinada por uma média móvel que visa um número médio de blocos por hora. Se os blocos forem gerados rápido demais, a dificuldade aumenta.

5. Rede

As etapas para operar a rede são as seguintes:

1. Novas transações são transmitidas a todos os nós.
2. Cada nó reúne novas transações em um bloco.
3. Cada nó trabalha para encontrar uma prova de trabalho difícil para seu bloco.
4. Quando um nó encontra uma prova de trabalho, ele transmite o bloco a todos os nós.
5. Os nós aceitam o bloco somente se todas as transações nele forem válidas e ainda não tiverem sido gastas.
6. Os nós expressam sua aceitação do bloco trabalhando na criação do próximo bloco da cadeia, usando o hash do bloco aceito como hash anterior.

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

As transmissões de novas transações não precisam necessariamente alcançar todos os nós. Enquanto alcançarem muitos nós, elas entrarão em um bloco em pouco tempo. As transmissões de blocos também toleram mensagens perdidas. Se um nó não receber um bloco, ele o solicitará quando receber o próximo bloco e perceber que perdeu um.

6. Incentivo

Por convenção, a primeira transação em um bloco é uma transação especial que cria uma nova moeda pertencente ao criador do bloco. Isso acrescenta um incentivo para que os nós apoiem a rede e fornece uma forma de distribuir inicialmente moedas em circulação, já que não há uma autoridade central para emiti-las. A adição constante de novas moedas é análoga aos mineradores de ouro que gastam recursos para adicionar ouro à circulação. Em nosso caso, o que se gasta é tempo de CPU e eletricidade.

O incentivo também pode ser financiado por taxas de transação. Se o valor de saída de uma transação for menor que seu valor de entrada, a diferença é uma taxa de transação adicionada ao valor de incentivo do bloco que contém a transação. Depois que um número predeterminado de moedas tiver entrado em circulação, o incentivo pode passar inteiramente para taxas de transação e tornar-se completamente livre de inflação.

O incentivo pode ajudar a encorajar os nós a permanecerem honestos. Se um atacante ganancioso for capaz de reunir mais poder computacional de CPU do que todos os nós honestos, ele terá de escolher entre usá-lo para fraudar pessoas, recuperando seus pagamentos, ou usá-lo para gerar novas moedas. Ele deve considerar mais lucrativo jogar pelas regras, regras que o favorecem com mais novas moedas do que todos os outros juntos, do que minar o sistema e a validade de sua própria riqueza.

7. Recuperação de Espaço em Disco

Depois que a transação mais recente de uma moeda estiver enterrada sob blocos suficientes, as transações gastas anteriores a ela podem ser descartadas para economizar espaço em disco. Para facilitar isso sem quebrar o hash do bloco, as transações são organizadas por hash em uma Árvore de Merkle [7][2][5], com apenas a raiz incluída no hash do bloco. Blocos antigos podem então ser compactados ao cortar ramos da árvore. Os hashes internos não precisam ser armazenados.

> *Figura 4. O diagrama original mostra transações organizadas em uma Árvore de Merkle e a poda de Tx0-Tx2 do bloco, mantendo o cabeçalho do bloco e a raiz de Merkle.*

Um cabeçalho de bloco sem transações teria 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. Com sistemas de computador normalmente vendidos com 2 GB de RAM em 2008, e com a Lei de Moore prevendo crescimento atual de 1,2 GB por ano, o armazenamento não deve ser um problema, mesmo que os cabeçalhos dos blocos tenham de ser mantidos em memória.

8. Verificação Simplificada de Pagamento

É possível verificar pagamentos sem executar um nó completo da rede. Um usuário precisa apenas manter uma cópia dos cabeçalhos dos blocos da cadeia de prova de trabalho mais longa, que ele pode obter consultando nós da rede até se convencer de que possui a cadeia mais longa, e obter o ramo de Merkle que liga a transação ao bloco em que ela foi carimbada no tempo. Ele não consegue verificar a transação por si mesmo, mas, ao ligá-la a um ponto da cadeia, pode ver que um nó da rede a aceitou, e os blocos adicionados depois dela confirmam ainda mais que a rede a aceitou.

> *Figura 5. O diagrama original mostra a cadeia mais longa de prova de trabalho, cabeçalhos de bloco e um ramo de Merkle para Tx3.*

Dessa forma, a verificação é confiável enquanto nós honestos controlarem a rede, mas é mais vulnerável se a rede for dominada por um atacante. Embora os nós da rede possam verificar transações por si mesmos, o método simplificado pode ser enganado pelas transações fabricadas de um atacante enquanto ele conseguir continuar dominando a rede. Uma estratégia para se proteger contra isso seria aceitar alertas dos nós da rede quando eles detectarem um bloco inválido, levando o software do usuário a baixar o bloco completo e as transações alertadas para confirmar a inconsistência. Empresas que recebem pagamentos frequentes provavelmente ainda desejarão executar seus próprios nós para obter segurança mais independente e verificação mais rápida.

9. Combinação e Divisão de Valor

Embora fosse possível lidar com moedas individualmente, seria incômodo fazer uma transação separada para cada centavo em uma transferência. Para permitir que o valor seja dividido e combinado, as transações contêm múltiplas entradas e saídas. Normalmente haverá uma única entrada proveniente de uma transação anterior maior, ou múltiplas entradas combinando quantias menores, e no máximo duas saídas: uma para o pagamento e outra devolvendo o troco, se houver, ao remetente.

Deve-se observar que a ramificação de saída, em que uma transação depende de várias transações, e essas transações dependem de muitas outras, não é um problema aqui. Nunca há necessidade de extrair uma cópia autônoma completa do histórico de uma transação.

10. Privacidade

O modelo bancário tradicional alcança certo nível de privacidade ao limitar o acesso às informações às partes envolvidas e ao terceiro confiável. A necessidade de anunciar todas as transações publicamente impede esse método, mas a privacidade ainda pode ser mantida ao interromper o fluxo de informações em outro ponto: mantendo as chaves públicas anônimas. O público pode ver que alguém está enviando uma quantia para outra pessoa, mas sem informações que vinculem a transação a alguém. Isso é

semelhante ao nível de informação divulgado pelas bolsas de valores, nas quais o horário e o tamanho das negociações individuais, a “fita”, são tornados públicos, mas sem informar quem eram as partes.

Figura 6. O diagrama original compara o modelo tradicional de privacidade — identidades, transações e terceiro confiável — com o novo modelo de privacidade, no qual identidades públicas são separadas das transações públicas.

Como uma camada adicional de proteção, um novo par de chaves deve ser usado em cada transação para impedir que elas sejam vinculadas a um proprietário comum. Alguma vinculação ainda é inevitável em transações com múltiplas entradas, que necessariamente revelam que suas entradas pertenciam ao mesmo proprietário. O risco é que, se o proprietário de uma chave for revelado, a vinculação poderá revelar outras transações pertencentes ao mesmo proprietário.

11. Cálculos

Consideramos o cenário de um atacante tentando gerar uma cadeia alternativa mais rapidamente do que a cadeia honesta. Mesmo que isso seja realizado, o sistema não fica aberto a mudanças arbitrárias, como criar valor do nada ou tomar dinheiro que nunca pertenceu ao atacante. Os nós não aceitarão uma transação inválida como pagamento, e os nós honestos nunca aceitarão um bloco que as contenha. Um atacante pode apenas tentar alterar uma de suas próprias transações para recuperar dinheiro que gastou recentemente.

A corrida entre a cadeia honesta e a cadeia de um atacante pode ser caracterizada como uma Caminhada Aleatória Binomial. O evento de sucesso é a cadeia honesta ser estendida por um bloco, aumentando sua liderança em +1; o evento de falha é a cadeia do atacante ser estendida por um bloco, reduzindo a diferença em -1.

A probabilidade de um atacante alcançar a cadeia a partir de um déficit dado é análoga a um problema de Ruína do Jogador. Suponha que um jogador com crédito ilimitado comece em déficit e jogue um número potencialmente infinito de tentativas para tentar alcançar o ponto de equilíbrio. Podemos calcular a probabilidade de ele algum dia alcançar esse ponto de equilíbrio, ou de um atacante algum dia alcançar a cadeia honesta, da seguinte forma [8]:

```
p = probabilidade de um nó honesto encontrar o próximo bloco
q = probabilidade de o atacante encontrar o próximo bloco
q_z = probabilidade de o atacante algum dia alcançar a cadeia a partir de z blocos atrás


q_z = {
  1,          se p <= q
  (q / p)^z, se p > q
}
```

Dada nossa suposição de que p > q, a probabilidade cai exponencialmente conforme aumenta o número de blocos que o atacante precisa alcançar. Com as chances contra ele, se não conseguir um avanço sortudo logo no início, suas chances se tornam praticamente nulas à medida que ele fica mais para trás.

Agora consideramos por quanto tempo o recebedor de uma nova transação precisa esperar antes de estar suficientemente certo de que o remetente não pode alterar a transação. Presumimos que o remetente é um atacante que quer fazer o recebedor acreditar que foi pago por algum tempo e, depois, mudar a transação para pagar de volta a si mesmo após algum tempo ter passado. O recebedor será alertado quando isso acontecer, mas o remetente espera que seja tarde demais.

O recebedor gera um novo par de chaves e fornece a chave pública ao remetente pouco antes da assinatura. Isso impede que o remetente prepare uma cadeia de blocos com antecedência, trabalhando nela continuamente até ter a sorte de avançar o suficiente e então executar a transação naquele momento. Depois que a transação é enviada, o remetente desonesto começa a trabalhar em segredo em uma cadeia paralela que contém uma versão alternativa de sua transação.

O recebedor espera até que a transação tenha sido adicionada a um bloco e que z blocos tenham sido encadeados depois dela. Ele não sabe a quantidade exata de progresso que o atacante fez, mas, supondo que os blocos honestos levaram o tempo médio esperado por bloco, o progresso potencial do atacante terá uma distribuição de Poisson com valor esperado:

```
lambda = z * q / p
```

Para obter a probabilidade de que o atacante ainda consiga alcançar a cadeia agora, multiplicamos a densidade de Poisson para cada quantidade de progresso que ele poderia ter feito pela probabilidade de que ele consiga alcançar a cadeia a partir daquele ponto:

```
sum_{k=0}^{infinito} (lambda^k * e^(-lambda) / k!) * {
  (q / p)^(z - k), se k <= z
  1,               se k > z
}
```

Reorganizando para evitar somar a cauda infinita da distribuição:

```
1 - sum_{k=0}^{z} (lambda^k * e^(-lambda) / k!) * (1 - (q / p)^(z - k))
```

Convertendo para 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 com 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 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

Propusemos um sistema para transações eletrônicas sem depender de confiança. Começamos com a estrutura usual de moedas feitas de assinaturas digitais, que fornece forte controle de propriedade, mas é incompleta sem uma forma de impedir o gasto duplo. Para resolver isso, propusemos uma rede ponto a ponto que usa prova de trabalho para registrar uma história pública de transações, a qual rapidamente se torna computacionalmente impraticável de alterar por um atacante caso os nós honestos controlem a maioria do poder computacional de CPU. A rede é robusta em sua simplicidade não estruturada. Os nós trabalham todos ao mesmo tempo com pouca coordenação. Eles não precisam ser identificados, pois as mensagens não são roteadas para um local específico e só precisam ser entregues com base no melhor esforço. Os nós podem sair e retornar à rede à vontade, aceitando a cadeia de prova de trabalho como prova do que aconteceu enquanto estiveram ausentes. Eles votam com seu poder computacional de CPU, expressando sua aceitação de blocos válidos ao trabalhar para estendê-los e rejeitando blocos inválidos ao se recusarem a trabalhar sobre eles. Quaisquer regras e incentivos necessários podem ser aplicados por meio desse mecanismo de consenso.

Referências

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

[2] H. Massias, X. S. Avila e J.-J. Quisquater, “Design of a secure timestamping service with minimal trust requirements,” em 20th Symposium on Information Theory in the Benelux, maio de 1999.

[3] S. Haber e W. S. Stornetta, “How to time-stamp a digital document,” Journal of Cryptology, vol. 3, n.º 2, páginas 99-111, 1991.

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

[5] S. Haber e W. S. Stornetta, “Secure names for bit-strings,” em Proceedings of the 4th ACM Conference on Computer and Communications Security, páginas 28-35, abril de 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,” em Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society, páginas 122-133, abril de 1980.

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