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

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

Traduzido por: Eduardo Abreu
www.mestretrader.com

Prefácio. Uma versão puramente ponto-a-ponto (P2P) de dinheiro eletrônico permitiria o envio de pagamentos online diretamente de uma pessoa para outra sem ter que passar por uma instituição financeira. Assinaturas digitais fornecem parte da solução, mas os principais benefícios são perdidos se um intermediário confiável ainda for necessário para prevenir gasto duplo. Nós propomos uma solução para o problema de gasto duplo utilizando uma rede ponto-a-ponto.

A rede registra data e hora das transações transformando as em um hash em uma corrente contínua de prova de trabalho codificada, formando um registro que não pode ser alterado sem que a prova de trabalho seja refeita. A corrente mais longa serve não somente como prova da sequência dos eventos testemunhados, mas prova de que eles vieram de um conjunto maior de poder de CPU. Enquanto a maior parte do poder de processamento for controlado por "*nós*" (pontos) que não estão cooperando para atacar a rede, eles irão gerar a corrente mais longa e superar os invasores. A rede em si requer uma estrutura mínima. Mensagens são transmitidas na base do melhor esforço, e os "*nós*" podem sair e se juntar novamente a rede à vontade, aceitando a corrente com a maior "prova de trabalho" como prova do que aconteceu enquanto eles estiveram fora.

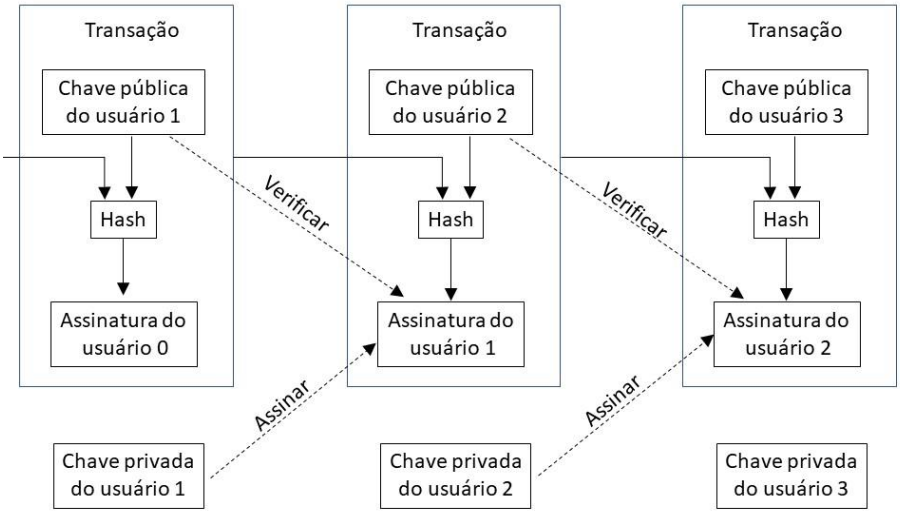
1. Introdução

O comércio pela Internet veio a depender quase que exclusivamente de instituições financeiras servindo como intermediários confiáveis para processar pagamentos eletrônicos. Enquanto o sistema funciona bem o suficiente para grande parte das transações, ainda assim ele sofre da fraqueza inerente desse modelo baseado em confiança. Na realidade, transações completamente irreversíveis não são possíveis, já que as instituições financeiras não podem evitar a mediação de disputas. O custo desta mediação aumenta os custos transacionais, limitando o valor mínimo para uma transação ser viável e eliminando a possibilidade de transações pequenas e casuais, e há um custo maior na perda da habilidade de fazer pagamentos não-estornáveis para serviços não-estornáveis. Com a possibilidade do estorno, a necessidade de confiança se torna ainda maior. Comerciantes devem desconfiar de seus clientes, incomodando-os por mais informações do que seriam realmente necessárias. Um certo percentual de fraude é aceita como inevitável. Estes custos e incertezas de pagamentos podem ser evitados pessoalmente usando uma moeda física, mas não existe mecanismo que faça pagamentos em canais de comunicação sem um intermediário confiável.

O que é preciso é um sistema de pagamentos eletrônicos baseado em provas criptográficas no lugar de confiança, que permita que duas partes interessadas em fazer transações diretamente, as façam sem a necessidade de um intermediário confiável. Transações que são computacionalmente impraticáveis de reverter protegeriam vendedores de fraude, e mecanismos de garantia poderiam ser facilmente implementados para proteger os compradores. Neste artigo, nós propomos uma solução para o problema do gasto duplo usando um servidor de "carimbos de tempo" (timestamp) distribuído ponto-a-ponto para gerar uma prova computacional da ordem cronológica das transações. O sistema é seguro desde que os *nós* honestos coletivamente controlem mais poder de processamento do que qualquer grupo cooperante de *nós* do invasor.

2. Transações

Nós definimos uma moeda eletrônica como uma cadeia de assinaturas digitais. Cada proprietário transfere a moeda para o próximo, assinando digitalmente um *hash* das transações anteriores e a chave pública do próximo proprietário e as adicionando ao fim da moeda. Um beneficiário pode conferir as assinaturas para verificar a cadeia de propriedade.

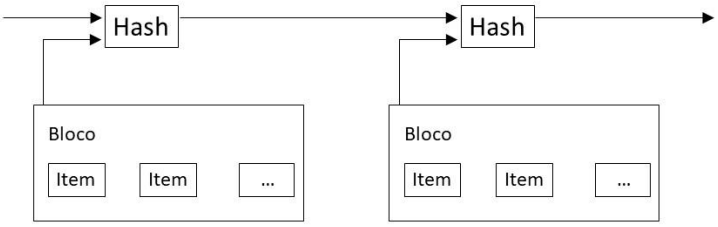


O problema, obviamente, é que o beneficiário não pode verificar se um dos proprietários não gastou duas vezes a moeda. Uma solução comum é introduzir uma autoridade central confiável, ou uma "Casa da Moeda", que verificaria cada transação por gasto duplo. Após cada transação, a moeda seria retornada a Casa da Moeda que emitiria então uma nova moeda, e apenas moedas emitidas diretamente da Casa da Moeda seriam confiáveis de não terem sido gastas duas vezes. O problema com esta solução é que o destino de todo o sistema monetário dependeria da empresa que toma conta da Casa da Moeda, com toda transação tendo que passar por ela, exatamente como um banco.

Nós precisamos de uma forma na qual o beneficiário saiba que os donos anteriores não assinaram nenhuma transação prévia. Para nosso propósito, a transação mais antiga é a transação que conta, então nós não nos importamos com tentativas subsequentes de gasto duplicado. A única forma de confirmar a ausência de uma transação é estar ciente de todas as transações. Em um modelo baseado em uma Casa da Moeda, a casa estava ciente de todas as transações e decidia qual delas chegou primeiro. Para que isso seja feito sem um intermediário confiável, as transações precisam ser publicamente anunciadas [1], e nós precisamos de um sistema para os participantes concordarem em um único histórico da ordem na qual elas foram recebidas. O beneficiário precisa de prova que na hora de cada transação, a maioria dos *nós* concordaram que ela foi a primeira recebida.

3. Servidor de Carimbos de tempo

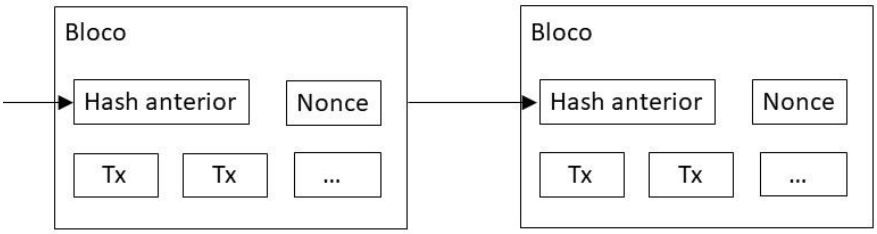
A solução que nós propomos começa com um servidor de carimbos de tempo. Um servidor de carimbos de tempo funciona pegando um *hash* de um bloco de itens para serem carimbados e publicando amplamente este *hash*, assim como em um jornal ou um post na Usenet [2-5]. O carimbo de tempo prova que os dados precisam ter existido naquele momento, obviamente, para que sejam incluídos no *hash*. Cada carimbo de tempo inclui o carimbo de tempo anterior em seu *hash*, formando uma corrente, com cada carimbo de tempo adicional reforçando os anteriores.



4. Prova de trabalho

Para implementar um servidor de carimbo de tempo distribuído em uma base ponto-a-ponto, nós vamos precisar utilizar um sistema de prova de trabalho similar ao *Adam Back's' Hashcash* [6], em vez de um jornal ou um post da Usenet. A prova de trabalho envolve procurar por um valor que quando transformado em hash, como com SHA-256, o *hash* começa com um número de zero bits. O trabalho médio necessário é exponencial no número de zero bits necessários e pode ser verificado executando um único *hash*.

Para a nossa rede de carimbos de tempo, nós implementamos a prova de trabalho incrementando um *nonce* no bloco até que um valor que dê ao *hash* do bloco a quantidade requerida de zero bits seja encontrado. Uma vez que o esforço da CPU foi gasto para satisfazer a prova de trabalho, o bloco não pode ser modificado sem refazer o trabalho. Como os blocos posteriores são acorrentados em logo em seguida, o trabalho para modificar o bloco inclui refazer todos os blocos após este.



A prova de trabalho também resolve o problema de determinar representação na tomada de decisão da maioria. Se a maioria estivesse localizada em "um-endereço-IP-um-voto", ela poderia ser subvertida por qualquer um capaz de alocar muitos IPs. Prova de trabalho é essencialmente "uma-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 maior parte do poder de processamento é controlado por *nós* honestos, a corrente honesta irá crescer mais rapidamente e ultrapassar qualquer corrente competidora. Para modificar um bloco antigo, um invasor teria que refazer a prova de trabalho do bloco e de todos os blocos posteriores e então recuperar o atraso e ultrapassar o trabalho dos *nós* honestos. Mostraremos adiante que a probabilidade de um invasor mais lento recuperar o atraso diminui exponencialmente, à medida que blocos subsequentes são adicionados.

Para compensar o aumento da velocidade do hardware e o interesse variável em rodar os *nós* com o passar do tempo, a dificuldade da prova de trabalho é determinada por uma média móvel visando um número de blocos por hora. Se os blocos forem gerados muito rápido, a dificuldade aumenta.

5. Rede

Os passos para executar a rede são os seguintes:

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

Nós sempre consideram a maior corrente como a corrente correta e vão continuar trabalhando em estendê-la. Se dois *nós* transmitem versões diferentes do mesmo bloco simultaneamente, alguns nós vão receber um ou outro primeiro. Neste caso, eles trabalham no primeiro que receberam, mas vão salvar a outra ramificação no caso dela se tornar a maior. O impasse será solucionado quando a próxima prova de trabalho for encontrada e uma das ramificações se tornar maior; os nós que estavam trabalhando na outra ramificação irão então mudar para a mais longa.

Transmissões de novas transações não necessariamente precisam alcançar todos os *nós*. Desde que elas alcancem muitos *nós*, elas vão entrar em algum bloco em breve. As transmissões de blocos também são tolerantes a mensagens perdidas. Se um *nó* não receber um bloco, ele irá solicitá-lo quando receber o próximo bloco e perceber que faltou um.

6. Incentivo

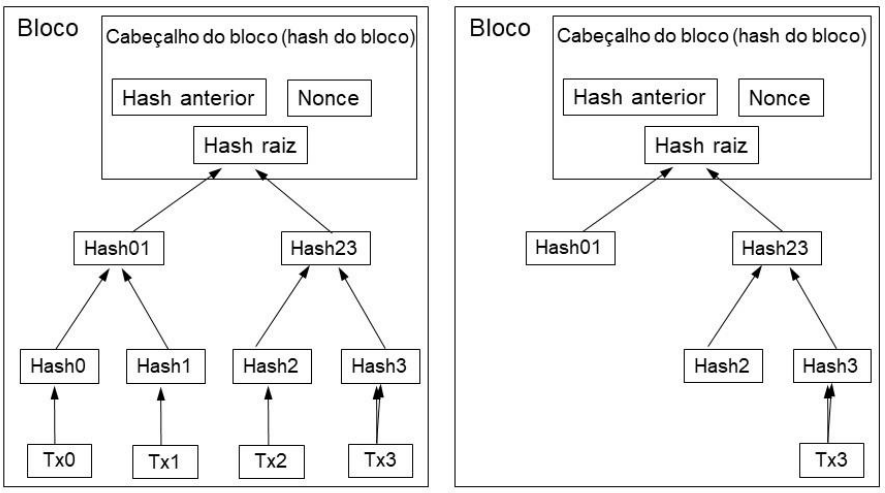
Por convenção, a primeira transação de um bloco é uma transação especial que inicia uma nova moeda de propriedade do criador do bloco. Isso adiciona um incentivo para os *nós* suportarem a rede, e estabelece uma forma de inicialmente distribuir moedas em circulação, já que não existe uma autoridade central para emitir essas moedas. A adição regular de uma constante de quantidade de novas moedas é análoga a dos mineradores de ouro gastando recursos para colocar mais ouro em circulação. No nosso caso, tempo de CPU e eletricidade que são gastos.

O incentivo também pode ser financiado com taxas de transação. Se o valor de saída de uma transação é menor que o valor de entrada, a diferença é uma taxa de transação que é adicionada ao valor de incentivo do bloco contendo a transação. Uma vez que um número pré-determinado de moedas estiver entrado em circulação, o incentivo pode mudar integralmente para taxas de transação e estar completamente livre de inflação.

O incentivo pode ajudar encorajar os *nós* a permanecerem honestos. Se um invasor ganancioso é capaz de reunir mais poder de processamento do que todos os *nós* honestos, ele teria que escolher entre usar esse poder para fraudar as pessoas roubando de volta os pagamentos, ou usá-lo para gerar novas moedas. Ele deverá achar mais lucrativo trabalhar conforme as regras, já que elas vão lhe gerar mais moedas do que todos os outros juntos, do que prejudicar o sistema e a validade de sua própria riqueza.

7. Recuperando espaço em disco

Uma vez que a última transação de uma moeda é enterrada em blocos o suficiente, as transações gastas anteriormente podem ser descartadas para salvar espaço em disco. Para facilitar esse ato sem quebrar o *hash* do bloco, transações são transformadas em hash em uma *árvore de Merkle* [7][2][5], apenas com a raiz incluída no *hash* dos blocos. Os *hashes* interiores não precisam ser armazenados.



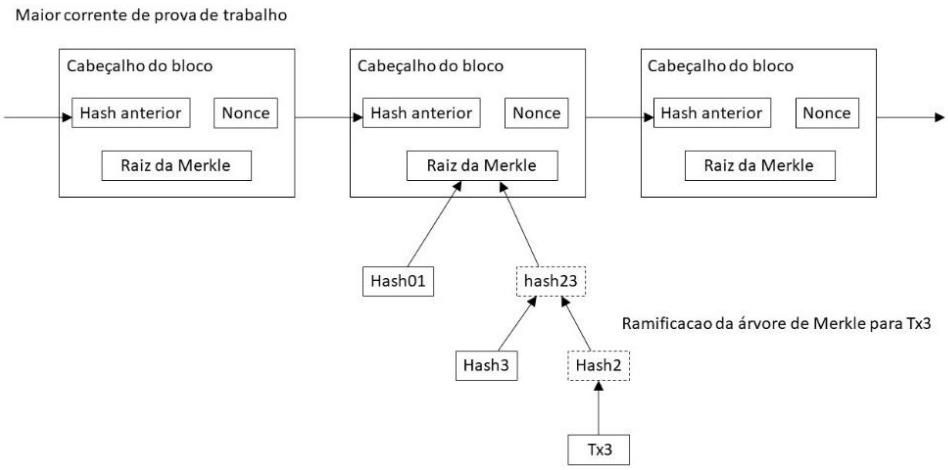
Transações em *hash* em uma arvore de Merkle

Após "podar" Tx0-2 do bloco

Um cabeçalho de bloco sem transações deve ter em torno de 80 bytes. Se supormos que os blocos são gerados a cada 10 minutos, 80 bytes * 6 * 24 * 365 = 4.2MB por ano. Com os sistemas de computador típicos sendo vendidos em 2008 com 2GB de RAM, e a lei de Moore prevendo um aumento de 1.2GB por ano, armazenamento não deve ser um problema mesmo se os cabeçalhos dos blocos devem ser mantidos na memória.

8. Verificação simplificada de pagamento

É possível verificar pagamentos sem ter que rodar um *nó* de rede inteiro. Um usuário precisa apenas manter uma cópia dos cabeçalhos dos blocos da maior corrente de prova de trabalho, que ele pode obter consultando os *nós* da rede até se convencer de que ele tem a maior corrente, e obter a ramificação da *arvore de Merkle* ligando a transação ao bloco onde ela está carimbada. Ele não pode verificar a transação para ele mesmo, mas ao liga-la a um local na corrente, ele pode ver que um nó aceitou a transação, e blocos adicionados após este irão confirmar ainda mais que a rede a aceitou.

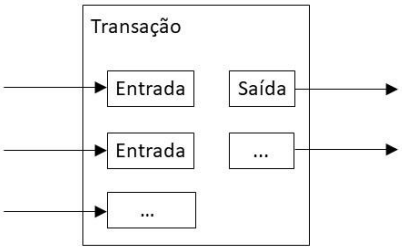


Como sempre, a verificação é confiável desde que os nós honestos controlem a rede, mas é mais vulnerável se a rede for dominada por um invasor. Enquanto os *nós* da rede podem verificar transações por si próprios, o método simplificado pode ser enganado por uma transação fabricada por um invasor enquanto o invasor continuar a dominar a rede. Uma estratégia para proteger-se contra este problema seria aceitar

alertas dos *nós* da rede quando eles detectam um bloco inválido, induzindo o software do usuário a baixar o bloco completo e as transações alertadas para confirmar a inconsistência. Negócios que recebem pagamentos frequentes vão provavelmente querer rodar seus próprios nós para uma segurança mais independente e verificações mais rápidas.

9. Combinando e dividindo valor

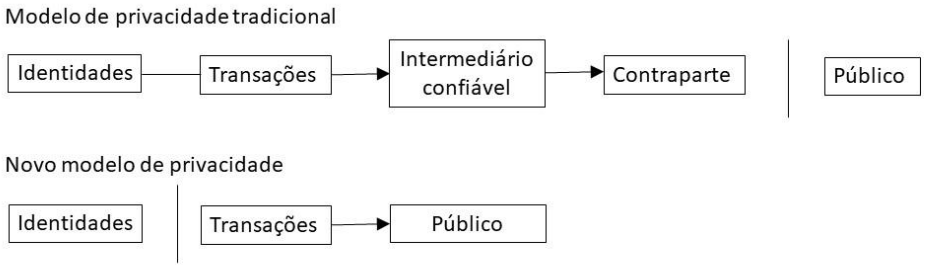
Embora seja possível manipular as moedas individualmente, seria complicado fazer uma transação separada para cada centavo em uma transferência. Para permitir que o valor seja dividido e combinado, transações contém múltiplas entradas e saídas. Normalmente haverá ou uma única entrada de uma grande transação anterior ou múltiplas entradas combinando pequenos valores, e no máximo duas saídas: Uma para o pagamento e uma para devolução de troco, se houver, de volta para o remetente.



Deve-se notar que dispersão, onde uma transação depende de várias transações, e estas transações dependem de muitas outras, não é um problema aqui. Nunca haverá a necessidade de extrair uma cópia independente 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 intermediário confiável. A necessidade de anunciar todas as transações publicamente inviabiliza este método, mas a privacidade ainda pode ser mantida quebrando o fluxo da informação em outro local: mantendo as chaves públicas anônimas. O público pode ver que alguém está enviando um montante de dinheiro para outra pessoa, mas sem informações ligando a transação a qualquer pessoa. Isso é parecido com o nível de informação liberado no mercado de ações, onde o tempo e o tamanho das vendas individuais, a "fita", são tornados públicos, porém sem dizer quem são as partes envolvidas.



Como um firewall adicional, um novo par de chaves deve ser usado para cada transação para evitar que sejam associadas a um proprietário comum. Alguma associação ainda é inevitável com transações de muitas entradas, que necessariamente revelam que suas entradas pertenciam ao mesmo proprietário. O risco é que se o proprietário de uma chave for revelado, associações poderiam revelar outras transações que pertenciam ao mesmo proprietário.

11. Cálculos

Nós consideramos o cenário de um invasor tentando gerar uma corrente alternativa mais rápido do que a corrente honesta. Mesmo que isso seja bem-sucedido, não irá tornar o sistema aberto para mudanças arbitrárias, como criar valor do nada ou tomar dinheiro que nunca pertenceu ao invasor. *Nós* não irão aceitar uma transação inválida como pagamento, e nós honestos nunca aceitarão um bloco contendo-as. Um invasor pode apenas tentar modificar uma de suas próprias transações para pegar de volta o dinheiro que ele gastou recentemente.

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

A probabilidade de um invasor alcançar de um determinado déficit é análoga ao problema da Ruina do Apostador. Suponha que um apostador com créditos ilimitados comece em déficit e jogue potencialmente um infinito número de vezes para tentar empatar. Nós vamos calcular a probabilidade de ele nunca conseguir isso, ou que um invasor simplesmente alcance a corrente honesta, como segue [8]:

p = probabilidade de um nó honesto encontrar o próximo bloco

q = probabilidade de um invasor encontrar o próximo bloco

q_z = probabilidade algum dia o invasor alcançar estando z blocos atrás

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

Assumindo que p > q, a probabilidade cai exponencialmente a medida que o número de blocos que o invasor tem que alcançar aumenta. Com a probabilidade contra ele, se ele não levar sorte logo no início, suas chances vão desaparecendo e ele acaba ficando muito atrasado.

Nós agora consideramos quanto tempo o destinatário de uma nova transação precisa esperar antes que seja suficientemente seguro de que o pagador não poderá alterar a transação. Nós assumimos que o remetente seja um invasor que quer fazer seu destinatário acreditar que ele o pagou por algum tempo, então mudar a transação para receber o dinheiro de volta depois que algum tempo se passou. O destinatário vai ser alertado quando isso acontecer, mas o remetente espera que seja tarde demais.

O destinatário gera um novo par de chaves e dá a chave pública para o remetente logo antes de assinar. Isso previne que o remetente prepare uma cadeia de blocos antes do tempo trabalhando nele continuamente até que ele seja sortudo o suficiente para avançar o suficiente a frente, então executando a transação naquele momento. Uma vez que a transação seja enviada, o remetente desonesto começa a trabalhar em segredo em uma cadeia paralela contendo uma versão alternativa da transação.

O destinatário aguarda até que a transação seja adicionada ao bloco e z blocos já tenham sido ligados após ele. Ele não sabe a quantidade exata do progresso que o invasor fez, mas assumindo que os blocos honestos tomaram a média esperada de tempo por bloco, o progresso potencial do invasor sera uma distribuição de Poisson com o valor esperado:

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

Para obter a probabilidade de um invasor ainda alcançar neste momento, nós multiplicamos a densidade de Poisson por cada quantidade de progresso que ele poderia ter feito pela probabilidade de ele conseguir alcançar a partir daquele ponto:

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

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

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

Convertendo em 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
```

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

Nós propusemos um sistema de transações eletrônicas sem depender de confiança. Nós começamos com a estrutura comum de moedas feitas a partir de assinaturas digitais, que oferecem forte controle de propriedade, mas é incompleta sem um sistema de prevenção de gasto duplo. Para resolver isso, nós propusemos uma rede p2p usando provas de trabalho para registrar um histórico público de transações que rapidamente se torna computacionalmente imprático para um invasor modificar se os nós honestos controlarem a maior parte do poder de processamento. A rede é robusta em sua simplicidade não estruturada. *Nós* trabalham todos ao mesmo tempo com pouca coordenação. Eles não precisam ser identificados, já que as mensagem não são encaminhadas para nenhum local em particular e apenas precisam ser entregues na base da "melhor esforço". *Nós* podem sair e voltar a rede a vontade, aceitando a cadeia de prova de trabalho como prova do que aconteceu enquanto ele esteve fora. Eles votam com seu poder de processamento, expressando sua aceitação dos blocos válidos ao trabalhar em estende-los e rejeitando blocos inválidos ao recusar a trabalhar neles. Quaisquer regras e incentivos necessários podem ser forçados 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 timestamping," 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.