Bitcoin: A Homie-to-Homie Digital Cash Vibe

Satoshi THE GOAT Nakamoto
satoshin@gmx.com
www.BitcoinForGenZ.org

Abstract. Yo fam, ever wish you could send cash to your crew without any bank drama or middleman vibes? Digital signatures are sick, but if you're still needing a trusted third party to stop double-spending, that's just sus. We've got a GOAT solution for this mess using a homie-to-homie network. Here's the tea: we timestamp transactions by hashing them into an endless chain of proof-of-work, creating a record that's locked down tight. You gotta redo all the proof-of-work to mess with it. The longest chain isn't just the ultimate proof of what went down; it's also proof that it came from the most CPU power flex. As long as the majority of CPU power is with the legit squad and not some shady crew trying to wreck things, they'll keep the chain lit and outpace the attackers. The network is mad chill with minimal structure, messages get broadcasted on a "best effort" basis, and nodes can dip in and out whenever they want. They just vibe with the longest proof-of-work chain to stay on track.

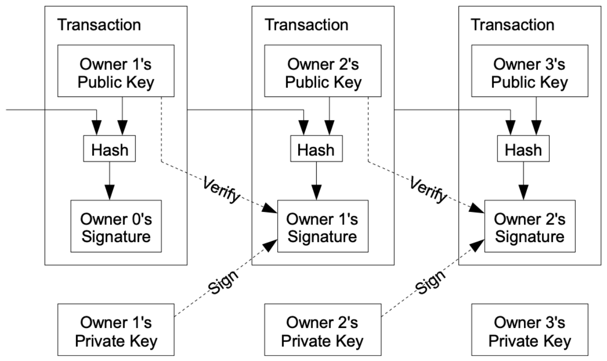
1. Introduction

Online cash and shopping usually need banks or payment apps as the ultimate middlemen. While it sorta works, it's super sus because you can't get rid of those trust issues. Transactions can't be un-reversible because these institutions will always need to play referee. That means fees are sky-high, small transactions are a no-go, and merchants have to watch their backs, asking for way too much info and accepting some fraud as just part of the game. Sure, you can dodge these issues with cash IRL, but no way to do that over the net without a trusted third party.

What we need is a digital cash vibe that's not about trust but about cryptographic proof. Imagine being able to send cash straight to your homies with zero middleman drama. Transactions would be so hard to reverse that sellers stay safe from fraud, and we could throw in some escrow for extra security. We're talking about solving the double-spending drama with a homie-to-homie distributed timestamp server that spits out hardcore proof of the transaction timeline. The system's lit as long as the honest crew has more CPU power than any shady hackers.

2. Transactions

Alright squad, here's the tea on electronic coins. Picture them as a chain of digital signatures, like an epic digital receipt. Every time someone passes the coin, they sign off on a hash of the previous transaction and the next person's public key, then slap that onto the coin. The payee (aka the lucky one getting the coin) can check these signatures to confirm the coin's ownership chain.



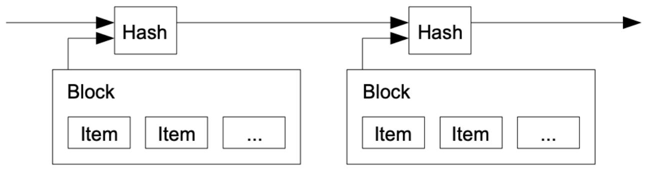
But here's the spicy part: the payee can't be 100% sure that someone didn't try to double-spend the coin earlier. Usually, you'd need a trust-fall buddy (like a mint) to keep things in check and stop any double-spend drama. In this setup, the coin has to bounce back to the mint after every transaction to get a fresh stamp, and only mint-approved coins are solid. But relying on the mint is basically like putting all your trust in a single bank, and that's kinda sus.

We need to level up so the payee can be totally sure no one tried to double-spend the coin earlier. The OG transaction is the real deal, so we don't care about any shady later attempts. To ditch the middleman, we're making all transactions public. Everyone needs to agree on the order of transactions to keep it lit. The payee needs proof that, at the time of each transaction, most of the nodes were vibing and agreed it was the first one. No cap.

3. Timestamp Server

Okay, fam, let's break down the timestamp server vibes. Think of it like a digital time-stamper. Here's the lowdown: you grab a hash of a bunch of stuff you want to timestamp, then blast that hash everywhere—like posting it on social media or sharing it in a mega group chat. This timestamp proves that your data was definitely in the game at that moment, because it's in the hash.

Here's the real flex: each new timestamp includes the hash of the last one, creating this sick chain of timestamps. It's like each new timestamp is backing up the ones before it, so if someone tries to mess with the past, they'd have to redo all the timestamps. It's like stacking receipts—each one supports the last, making it super hard to fake the whole thing. Total GOAT move.

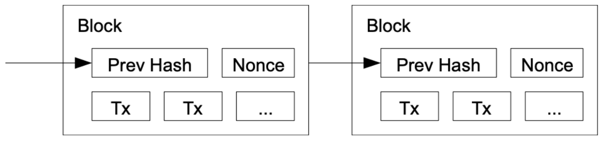


4. Proof-of-Work

Alright fam, if we wanna set up a timestamp server that's totally Homie-to-Homie, we gotta roll with Proof-of-Work, kinda like Adam Back's Hashcash—way better than posting on old-school forums or newspapers.

Proof-of-Work is basically a vibe where you hunt for a number that, when hashed (think SHA-256), starts with a bunch of zeroes. It's like an epic quest, and the more zeroes you need, the more CPU sweat you gotta pour in. The struggle is real, but you can check if it's legit with just one hash.

For our timestamp squad, we keep cranking a nonce in the block until we get a hash with those zeroes. Once the CPU flexes hard and nails it, the block's locked in. Messing with it later means redoing all that Proof-of-Work for that block and every block after it.



Proof-of-Work also solves the debate on who's in charge. If we went with one-IP-address-one-vote, it'd be easy to game—just grab tons of IPs. Instead, Proof-of-Work is like one-CPU-one-vote. The longest chain, with the most Proof-of-Work, is the real deal. If the majority of CPU power is with the honest squad, their chain will outgrow and outshine any shady ones. Trying to mess with a past block? You'd have to redo the Proof-of-Work for that block and every one after it, then catch up with the honest crew. Hint: The odds of catching up drop fast as new blocks keep stacking up.

To keep it all balanced with faster hardware and different node vibes, we adjust the Proof-of-Work difficulty based on how many blocks we wanna see per hour. If blocks drop too quickly, we ramp up the difficulty to keep things chill.

5. Network

Here's the vibe on how the network rolls:

1) Fresh transactions get blasted to all nodes, like dropping a new Insta story.
2) Each node grabs these transactions and packs them into a block, like making a new playlist.
3) Nodes work hard to solve a POW challenge for their block, kinda like grinding for a high score.
4) When a node nails it, it shares the block with everyone, like posting a banger on TikTok.
5) Nodes only accept the block if all transactions in it are legit and not double-dipped. No scams here.
6) Nodes build the next block using the hash from the accepted block, keeping the chain going.

Nodes always ride with the longest chain as the true OG and keep building on it. If two nodes drop different versions of the next block at the same time, some nodes might catch one version first. They'll work on that but keep the other version in their back pocket just in case it becomes the new hottest trend. The tie breaks when the next Proof-of-Work comes through and one chain gets longer; nodes on the shorter chain will switch to the longer one, like changing your profile pic when the new trend drops.

New transactions don’t need to hit every single node to get into a block. As long as they hit enough nodes, they’ll get into a block soon. Blocks get shared even if some messages ghost. If a node misses a block, it’ll just catch up and grab it when it gets the next block and realizes it’s been left out.

6. Incentive

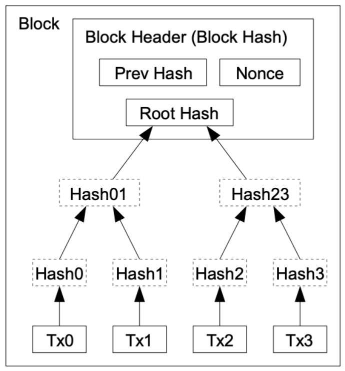
Here's the lowdown: the first transaction in each block is a VIP move that hands out a shiny new coin to the block creator. This setup is a total win for nodes because it gives them a solid reason to keep the network rolling and gets coins into the wild without needing a big boss to dish them out. It’s like how gold miners put in the grind to pull gold from the ground, but in our world, it’s all about burning CPU power and electricity.

On top of that, there’s the whole deal with transaction fees. If a transaction’s output is less than its input, the leftover is a fee that boosts the block’s reward. Once enough coins are out there, this reward can shift to being powered by transaction fees alone, keeping things inflation-free.

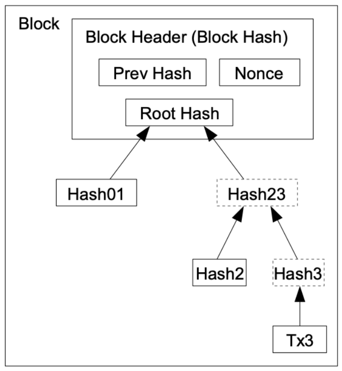
And here’s the kicker: this incentive system also keeps things on the up and up. If someone gathers more CPU power than all the honest nodes combined, they face a dilemma: use that power to cheat and steal payments or use it to mint new coins. Sticking to the rules and earning more coins honestly is way more rewarding than wrecking the system and risking their own stash.

7. Reclaiming Disk Space

Once the newest transaction in a coin gets buried under a ton of blocks, we can ditch those old transactions to save space. To do this without messing up the block’s hash, we use a Merkle Tree [7][2][5]—think of it like a digital family tree where only the root is in the block’s hash. We trim off the old branches, so only the essential bits stick around. No need to save the middle hashes.



Transactions Hashed in a Merkle Tree

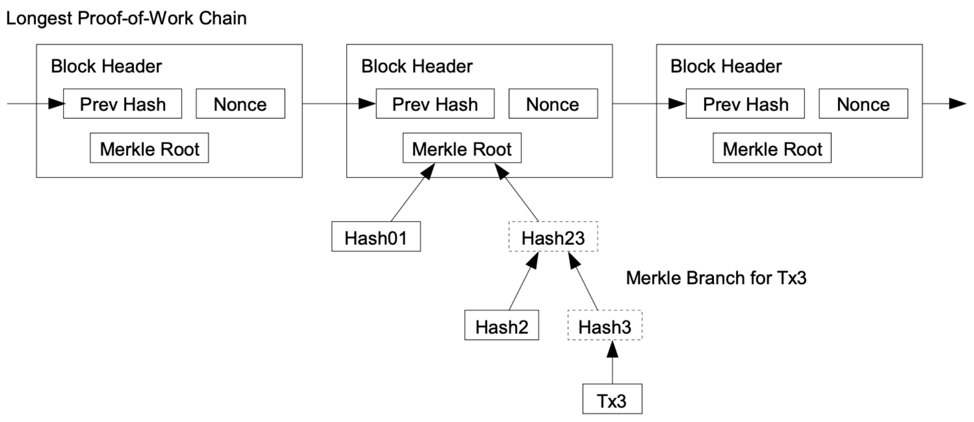


After Pruning Tx0-2 from the Block

A block header with no transactions is about 80 bytes. So, if blocks drop every 10 minutes, that’s 80 bytes * 6 * 24 * 365 = 4.2MB a year. With most computers packing 2GB of RAM as of 2008, and Moore's Law making RAM grow by 1.2GB a year, keeping these headers on lock shouldn’t be a big deal.

8. Simplified Payment Verification

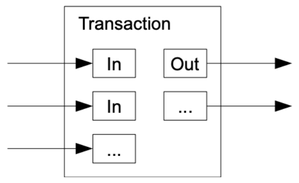
So, you don't need to be a full-on network node to verify payments—just keep a copy of the block headers from the longest proof-of-work chain. Hit up network nodes until you've snagged the longest chain and grab the Merkle branch linking your transaction to the block it's timestamped in. You won't be able to check the transaction yourself, but by tying it to the chain, you can see that the network has accepted it. More blocks popping up after it just makes it more solid.



Verification's pretty reliable as long as honest nodes are running the show, but if an attacker takes over, things get a bit sus. While full network nodes can do their own checks, this simpler method can be tricked if an attacker has enough juice to overpower the network. To stay on the safe side, you could set up alerts from network nodes that flag any invalid blocks. This way, your software gets the heads-up to download the full block and check out any shady transactions. Businesses that get payments on the reg might still want to roll with their own nodes for extra security and faster verification.

9. Combining and Splitting Value

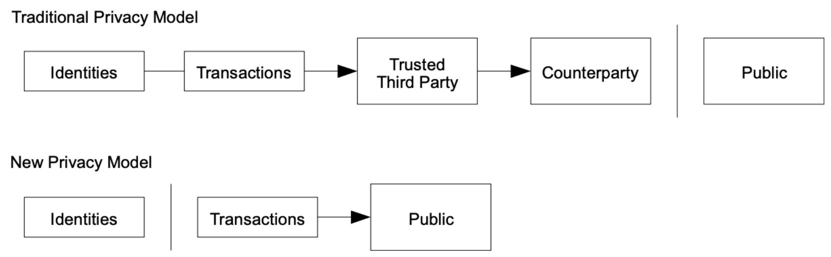
Handling coins one-by-one? Nah, that's a whole vibe killer. We're all about efficiency here. Instead of making a separate transfer for every single cent, transactions let you mix and match inputs and outputs like a pro. You can roll with one big input from a previous transaction or combine smaller amounts like stacking up your favorite snacks. Then, you've got up to two outputs: one for the actual payment and one for any change that gets sent back to you.



No stress about fan-out either—where a single transaction spawns a bunch of others. You don't need to dig through a full history for each move. It's all about keeping it chill and streamlined.

10. Privacy

Alright, so traditional banks keep things hush-hush by only letting the involved peeps and their trusted third party see the deets. But since we're all about that public transaction life, we gotta switch up the privacy game. Here's the trick: keep those public keys on the down-low. Sure, everyone can see that cash is moving from one spot to another, but without knowing who's behind it. Think of it like stock market vibes—everyone knows when and how much is traded, but the traders stay off the radar.



For extra stealth mode, swap out your key pair with each transaction. This way, no one can link your moves together like a social media feed. Just a heads-up, though—when you're dealing with multi-input transactions, it's kinda obvious that all the inputs come from the same source. If someone figures out who owns a key, they might piece together other transactions linked to that key. Keep it fresh and keep it sneaky!

11. Calculations

Imagine an attacker trying to outpace the honest chain with a sneaky alternate chain. Even if they manage to speed up, it doesn't mean they can just pull magic tricks like creating fake value or grabbing cash that's not theirs. Nodes aren't going to fall for dodgy transactions, and honest nodes will always reject blocks with them. The only thing an attacker can really do is mess with their own transactions to claw back cash they've spent.

The battle between the legit chain and the shady attacker chain is like a Binomial Random Walk. Winning means the honest chain just got a +1 lead by adding a new block, while losing means the attacker's chain got a +1 and closed the gap.

The chance of an attacker catching up from a deficit is kind of like a Gambler's Ruin scenario. Picture a gambler with infinite funds starting off behind and playing forever to break even. We can figure out the odds of the attacker catching up to the honest chain like this [8]:

p = probability an honest node finds the next block

q = probability the attacker finds the next block

qz = probability the attacker will ever catch up from z blocks

behind

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

So, if we’re vibing with p > q, the odds of our attacker catching up drop hard and fast as they lag behind more and more. If they don’t score a lucky break early on, their chances are basically zero. They’re toast if they keep falling behind!

Now, let’s break down how long the recipient should chill before they’re sure the sender isn’t about to pull a sneaky switcharoo. Imagine the sender is a shady scammer trying to trick the recipient into thinking they’ve been paid, only to flip it later and pay themselves instead. The recipient will get the tea if this goes down, but the scammer is hoping it'll be too late to do anything about it.

To avoid getting played, the recipient sets up a new key pair and slides the public key to the sender right before hitting send. This move stops the scammer from prepping a fake block chain ahead of time, grinding until they get a lucky break, and then switching up the transaction.

Once the transaction is sent, the shady sender starts working on a secret parallel chain with a fake version of the transaction. The recipient just has to chill until the transaction is locked in a block and z blocks stack up after it. They might not know exactly how much progress the scammer has made, but assuming the honest blocks are coming in at the usual rate, the attacker’s progress will follow a Poisson distribution with expected value:

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

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

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

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

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

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

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

q=0.1	
z=0	P=1.0000000
z=1	P=0.2045873
z=2	P=0.0509779
z=3	P=0.0131722
z=4	P=0.0034552
z=5	P=0.0009137
z=6	P=0.0002428
z=7	P=0.0000647
z=8	P=0.0000173
z=9	P=0.0000046
z=10	P=0.0000012

q=0.3	
z=0	P=1.0000000
z=5	P=0.1773523
z=10	P=0.0416605
z=15	P=0.0101008
z=20	P=0.0024804
z=25	P=0.0006132
z=30	P=0.0001522
z=35	P=0.0000379
z=40	P=0.0000095
z=45	P=0.0000024
z=50	P=0.0000006

Solving for P less than 0.1%...

P <	0.001
q=0.10	z=5
q=0.15	z=8
q=0.20	z=11
q=0.25	z=15
q=0.30	z=24
q=0.35	z=41
q=0.40	z=89
q=0.45	z=340

12. Conclusion

We’ve dropped a next-level system for cashing out online without all the trust drama. We kicked it off with the classic coins made from digital signatures—super tight for ownership control but kinda meh without stopping double-spending. To fix that, we’re all about a homie-to-homie squad using proof-of-work to keep a public transaction log that’s basically impossible for hackers to mess with if the legit nodes are flexing most of the CPU power.

Our network is a total boss at being both simple and rock-solid. Nodes sync up and grind together with minimal coordination—no need for fancy IDs or specific message routes. Messages just get tossed around on a “try your best” basis. Nodes can bounce in and out whenever they want, catching up on the proof-of-work chain to see what’s been going down while they were MIA. They use their CPU muscle to back valid blocks and ditch the sketchy ones. Everything runs smoothly with this consensus vibe—rules and incentives are enforced by the crew’s collective power.

References

[1]W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998. [2]H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3]S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4]D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5]S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7]R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

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