CHAPTER 7

NANOSECOND SUITCASE

What if the cypherpunks actually won? How would an anonymous digital infrastructure not be overwhelmed with spam, fraud, and forged digital cash? Some of the problems their cipher utopia was facing could be solved by a computational tool called *proof of work*. Exploring how this technology functions reveals a menagerie of experimental digital tokens and currencies—hashcash, RPOW, bit gold, b-money, and other Bitcoin precursors—and introduces the challenge of building secret banks.

WHAT IF WE WIN?

Adam Back made the T-shirts for the revolution. They featured blocks of white text on black cotton, including a warning, text of relevant laws and documents, four lines of code, and a big square of machine-readable barcode. The shirts were legally classified as munitions in the United States: you could not let a foreign national *see* the shirts, much less photograph or export one. Wearing one of Back's shirts on an international flight was a complex kind of crime. In France, wearing the shirt could accrue a massive fine and jail time. The code on the shirt was the RSA encryption algorithm—a working implementation of public key cryptography—rendered in the brutally laconic programming language Perl.

The shirts mocked the structure of the regulations by their very existence. So did the people who got RSA-in-Perl tattoos: able to say, along with martial artists in 1980s action movies, that their very bodies were classified as deadly weapons. Putting a shirt on, being photographed for a magazine or—worse still—appearing on television, was to suggest the impossibility of containing the cypherpunk toolkit and keeping it from widespread use. The garment implied victory.

Adam Back was then faced with the question implicit in May's Xth Column scheme: What if, in fact, the cypherpunks won?

Crypto for the millions! Public key encryption software becomes so widespread, reliable, and convenient that there is no reason to communicate insecurely. Your most casual online exchanges are authenticated by public key signatures, transacted over anonymous remailers, and wholly enciphered from outsiders. Governments effectively abandon cyberspace and the cypherpunk dream is realized.

It is immediately rendered useless by *spam*. The new crypto anarchist order blows out on the launch pad, overwhelmed with penis-enlargement promises, ads for counterfeit watches and home refinancing and deadstock appliances, porn-site hustles, phishing scams, and "Hello dear friend in Christ. I have Eighteen million five hundred thousand united states dollars fortune . . . ."

The most effective tools for keeping email spam traffic at manageable levels used identities and addresses (whitelisting) or the content of the messages themselves (filtering, whether based on keywords or ongoing machine learning). Wide adoption of encryption by individuals made the messages opaque to everyone but their intended recipients, so even the crudest filter—one that just looked for "porn" or "only $" to discard messages—became useless. The addition of tools like anonymous remailers, passing messages along without disclosing their original sender,

wiped out the utility of blocking mail from suspicious or known-bad addresses. What an inglorious fate for the cypherpunk dream: to succeed against the black glass monolith of the NSA and its army of top-tier mathematics PhDs only to be beaten by the small-time hucksters, pill touts, and con artists of the spam world, as though NASA somehow lost Mission Control to a Floridian time-share scam. Would the Other Plane be an endless wave of rip-offs, phishing messages, spoofs, spam, and hoaxes—an economy of messages flooded with worthless paper?

On March 28, 1997, Back presented his first draft of a postage system that could address this embarrassing scenario. What if the very computational work used to create and send an encrypted message—work that had become steadily more efficient for decades—could be turned against abuse of the encrypted network? To understand what Back built—and its consequences for digital cash—we must first understand what "computational work" meant.

NANOSECOND SUITCASE

Grace Hopper used to travel with a suitcase full of nanoseconds.

Meeting with students and generals, speaking with Congress, with engineers, or on television, she brought luggage filled with units of computational time for her audience to take home.[1] A computer scientist and one of the very first programmers, Hopper liked physical analogies: when she developed the first compiler, a program for transforming instructions written in programming language into machine language to be executed by a computer, she thought about the passing rules from when she'd played basketball—ways of "jumping" between the steps of a program.[2] She knew how hard it was to understand the *time* of computation and telecommunications, especially the

wasted time. It was difficult for humans to think in terms of tenths or hundredths of a second, much less millionths (micro) and billionths (nano). Men gleaming with military brass would ask, Why does a satellite transmission take so long? How can we build faster computers? And Hopper would reach into her bag.

Her nanosecond was a length of wire almost thirty centimeters long, 11.8 inches—the distance light travels in a vacuum in that time, the upper bound on any movement of information in the universe.[3] There are many nanoseconds, she would tell the admirals, between a ship at sea and a satellite in orbit; hence the delay. A computer with inches of wire between components was racking up the nanoseconds with each instruction and each result—a pulse of electricity, passing back and forth. (The Harvard Mark I, the first computer she worked on, had 530 miles of wiring.[4]) A badly designed or poorly programmed computer was wasting comparatively glacial microseconds, as Hopper would illustrate by holding one up: a massive coil of wire, 984 feet long. "I sometimes think we ought to hang one over every programmer's desk, or around their neck—so they know what they're throwing away when they throw away microseconds."[5]

This perspective can be dizzying: one clock tick of a high-end modern computer's processor (at three gigahertz) takes about a third of a nanosecond, during which it can execute some amount of work. If we imagine that tick as a full second—*one-Mississippi*—then the time it takes to send a packet of data one way from New York to San Francisco over a fiber optic cable, twenty-one microseconds, is the equivalent of about two years. Wagon trains waiting out the winter in Iowa or clipper ships sailing around Cape Horn and up the Chilean coast could beat that schedule—and twenty microseconds is a duration still

very far below the human ability to detect. That's the temporal scale of computing, the scale where Hopper worked, and where Back's proposal was set.

When you send an email to me, Back proposed, your email program generates a "hash" of the message—a small piece of data corresponding to the data of the whole of the email. The hashed data includes components like the date and time the message was sent and the receiver's address, so each hash is good for one and only one message. Making this hash takes some very small amount of computational work on your part. Because of the properties of a particular set of tools called *partial hash collision algorithms*, we can turn the dial on how much work it will take your computer to produce this valid hash.

Then, on the receiving end, my email program checks that the hash is correct. If the hash indeed corresponds to the message sent, I receive the message; if not, the message is discarded. The deep ingenuity of the notion kicks in with the fact that you and I, writing back and forth—even writing to mailing lists and the like—never notice that this is happening. The computational work happens too fast to matter.

However, if you start emailing people in very large numbers, in the hundreds of thousands, the work becomes onerous. Producing the correct hash for every single message becomes a problem only in aggregate, with your computer slowing to a grind as the fans whir to cool the overheating chips. Since most spammers only operate profitably at a scale of tens to hundreds of millions of messages, this creates a built-in brake on their ability to do business, bumping them down from wholesale to retail. In the long term, as the performance of new computers improves, the ability to dial up the difficulty of the hashing problems will let this system keep pace.

The hash accompanying the message therefore functions as a kind of metered postage—a small token of effort, an expenditure, which inhibits mass mailing while leaving personal correspondence effectively untouched. A little "proof of work," if you will. Prior to Back's announcement, proposals had been made advocating some kind of micropayment stamp, some small financial gesture or quantity of computing work, using a digital cash system.[6] This is why, despite this token bearing seemingly little relationship to what we might think of as money, Back called his concept *hashcash.*

He continued to refine the idea in the following years. What else could you do with this hash that operates as a small token of effort, as a proof of work, easy to do little and hard to do big? In a 2002 paper about hashcash, Back lists potential applications for the idea, concluding with: "hashcash as a minting mechanism for Wei Dai's b-money electronic cash proposal, an electronic cash scheme without a banking interface." In fact, hashing tools would be useful for minting money and creating banks in more ways than one.

DESTROY EVERY VESTIGE OF STRUCTURE

"Thus the concept of hashing finds wider application than just in computing addresses," wrote G. D. Knott in a survey of hashing functions in 1975. "It is a basic concept which can be useful in many circumstances."[7] Indeed it can. A *hash,* as in the random jumble of ingredients produced by hashing, cutting or chopping, began as the solution for a seemingly simple question with profound implications: What is the fastest way for a computer to look something up?

The data a program needs may be scattered across the space of available memory—those magnetic stripes laid on the

spinning platter of a hard disk, or on a reel of tape spooling and unspooling. Even a simple program will be making many small changes to whatever churns in its working memory. How does it find those places, redirect when it copies one part to another location, and return to a thing it has altered? As fast as the machinery can move, there is still travel time for the read/write head to find its place on disk, and as Grace Hopper would remind us, that time adds up. Either you update the whole table that lists the location of every item in memory, every time it changes, or you add to the table unsystematically and go through the whole thing again every time you need something.

A solution to this problem was "scatter storage," a way of making a key that could correspond to any given entry in storage—to where that data lives on the disk or the reel of tape—with a transformation that evenly distributes keys through the table of things to look up.[8] You are as likely to find what you need anywhere you land, if the distribution is really equal. This approach, pioneered by Hans Peter Luhn at IBM Poughkeepsie, works poorly for humans but wonderfully for computers. As Matthew Kirschenbaum summarizes it, in his study of the applications of hashing to computer memory and digital forensics, "structure—and with it predictable access routines for the drive's mechanical read head—emerged from normal patterns of statistical distribution among the numeric indices rather than from any kind of semantic correlation between index and key."[9] Or, in the beautiful phrase of a history of IBM's early computers: "[Luhn's] fundamental insight was to see merit in deliberately abusing keys, thereby attempting to destroy every vestige of structure."[10]

To do this, you need something rather magical: a way of transforming data that will always give the same result for the same data, and a different result for different data, so the same key

won't correspond to different inputs. (The term for this accidental correspondence—giving the same hashed key for different data—is a *collision.*) The magical transformation has become a commonplace matter in computer science: a *hash,* a function that takes data of any size and returns data of fixed size, usually much shorter, which corresponds to the original data. Any change to the original will produce a different hash. You can select a particular hashing algorithm and tune it to different parameters, producing brief units of gibberish that directly correspond to *what* that data is.

Hashing schemes and algorithms multiplied, as did the uses for hashing itself. Hashes could be used to confirm that two digital objects—texts, files of code, pieces of media—were precisely the same and had not been altered by corruption or an adversary's deliberate action. Furthermore, you could confirm that identity without having to compare whole objects, or even to reveal precisely what the objects are; instead you could just compare their hashes. You don't need to know the original texts to know that one is different. Finally, you cannot figure out the original data from the hash of the data. It is—at least in theory, if not always in practice—not *reversible.* The hash of the thing tells you nothing about the thing, except that the hash corresponds to it, and to it alone. If you run an online service that requires passwords, when your user logs in, their system can send the hash of their password to you, rather than the password itself. You can confirm that they have the secret, verified by the hash, without having it yourself—recognizable without being known.

We will take this technology further, to explain how it came to be built into the heart of digital cash, with two unorthodox applications of hashing tools. First, hashes can also be used to

create irrefutable chains of linked, time-stamped events—blocks of linked events, chained together (blockchains, if you will). Second, and bringing us back to Back and hashcash, hashes can be used to demand and verify an exact amount of work from a computer.

ENTROPY ARCHIVE

Consider an esoteric but vitally important problem: the challenge of distributing verifiable, reliable random numbers. We need and use reliable randomness for doing quality assurance checks on new cars and pharmaceuticals, for recounting ballots to ensure the integrity of a vote, for conducting medical screenings, even for generating the secret keys needed for encryption or making financial or military decisions that can't be predicted. With fake randomness you could manipulate a market, fix a lottery, produce an illusion of security with secret codes you know how to break, and hide all kinds of malfeasance. To meet this challenge, some organizations generate their randomness in-house. The network security firm Cloudflare keeps a wall of one hundred lava lamps in their San Francisco office. The fluid movement in the lamps is a high-contrast source of entropy, an estimated 16,384 bits worth, perfect for capture by a digital camera (with changes to the ambient light) whose images can be the seed for generating random numbers. (Cloudflare also uses the spinning of dual pendulums, the decay of a chunk of uranium, and other less entertaining industry-standard randomness sources.) But that's private, and subject to potential manipulation.

What about a public, shared, reliable source of randomness? How could you be sure you could trust the information? Imagine an adversary wants to falsify a set of random numbers to their

advantage. Our enemy means to plant the "random" factor that determines where we do a ballot review so it will be conducted in a preselected district to conceal a rigged election. This challenge, and its solution, will loom large for digital cash.

The National Institute for Standards and Technology (NIST) in the United States maintains a randomness beacon, a "public randomness service": a new string of random characters, 512 bits of entropy, generated every minute and broadcast on the Internet. They have been doing this since a little before noon on September 5, 2013; the first message began "17070B49D . . . ." If you are incorporating randomness into life-and-death decisions and processes, how can you be sure the latest string of random characters from NIST has not been inserted into their system by an enemy hacking into their website? Each new unit of entropy is signed with NIST's private key, just like the digital cash withdrawn from a bank using David Chaum's system—but perhaps your adversary has also stolen that key.

Each of NIST's initial random numbers are combined with some related information (a time stamp, a status code, and so on)—including the value of the *previous* randomness broadcast. This collection of data is then hashed all together. NIST signs that hash with its private key, hashes the whole thing again, and broadcasts the resulting string: "63C4B71D51 . . . ." The results of the hashing process are easy to verify as corresponding to the input data but impossible to predict in advance. It's here that the significance of including the previous broadcast comes in. Your enemy can steal NIST's key, and they can figure out how you'll use the randomness so they can cook up factors that will produce the outcome they want. But the randomness broadcast from NIST will have to include the prior broadcast, which anyone can check, making the results of the enemy's hashing impossible for

them to control. That prior broadcast's hash incorporates the broadcast before it, in turn, which incorporates the one before it, link by link in the chain, four years back, sixty seconds at a time. The hashes using previous hashes makes the latest broadcast reliable by connecting it to a public archive, with each event cryptographically incorporating previous events, so that attempts to change the past are immediately apparent—breaks in the chain.

COSTLY BITS

Consider one last thing you could do with hashes. Recall that different hashes generated by the same data are called *collisions*. Collisions are to be avoided if you're using hashes to look something up or verify passwords: a hashing algorithm that gives you a bunch of different hashes for the same input would be disastrous.

However, with such a system you could demand a particular hash out of the many possible hashes the algorithm can generate from some given data. If the algorithm can generate a lot of possible hashes from the input, you can request a hash with certain properties—that some number of its initial bits sum up to zero, for instance. You could make this request knowing precisely how hard it will be to find the correct hash output, the hash that meets your requirement and corresponds with the data, without knowing what the output itself would be in advance. With a particular hashing algorithm like SHA-1—used by Adam Back, Hal Finney, and the earliest drafts of Bitcoin—there are no shortcuts to producing the right hash. "Because of SHA-1's properties," Finney wrote, "the only way to find a string with a large collision size is by exhaustive search: trying one variation after

another, until you get lucky."[11] Here is an example of a hashcash token for one of Finney's emails:

"1:28:040727:halmail1@finney.org::1c6a5020f5ef5c75:63cca52"

The SHA-1 hash of this string looks like this: "0000000a86d41df172f177f4e7ec3907d4634b58"—with seven zeros. Someone's computer will have to produce and discard many hashes from the email Finney wrote before it finds that string with seven zeros at the beginning: about a million tries for a 20-bit collision, a billion for 30. (The Finney example is a 28-bit collision.) As with other kinds of hashes, it takes some work to produce each one, but it is trivial to verify that one is correct once you have it in hand.[12] By changing the properties you require from the hash of your data, you can make it *arbitrarily difficult* to compute the correct hash for something.

What use could we get from such an absurd machine? You could build a mechanism, like the Sphinx, that asks a riddle. There is only one right answer to the riddle. As the creator of the mechanism, you do not know the answer, but you know *exactly* how hard, and how time consuming, it will be to guess successfully. If people guess too quickly, you can crank the ratchet a few teeth forward into greater difficulty, demanding more "proof of work." With such a device, you could set the amount of work that was of interest to Adam Back: the computational time it would take to produce a hash of a particular email. A collision could be demanded that would be invisible to the everyday email correspondent but an impassable threshold for someone trying to mass-mail millions of people. Call this quantity of demonstrable work "postage," a digital object that was hard to make, and easily verified as having been hard to make.

The historian Anson Rabinbach's *The Human Motor* documented the search for quantifiable metrics for the expenditure of human force, and the "ergographic" instruments that served to measure and represent human bodily work and muscular energy. These instruments (special gloves, arrangements of dumbbells) were in the service of a larger project—to understand the nature of fatigue and find "nerve whips" that could overcome exhaustion and the diminution of bodily power—but reading Rabinbach's history now it's easy to imagine these systems as prototypes of minting mechanisms for a currency based on units of human effort. The project he documented was constantly undermined by the problem of confounding factors of measuring effort and fatigue: Was it muscles, or nerves, or keeping a fixed position, tedium, diet, or temperature?

With partial hash collision algorithms, this fantasy was realized—but for machines, not humans. The algorithms are an exquisitely precise way to demand and demonstrate quantities of computational work: cycles of a central processing unit, expressing watts of power consumed. Furthermore, being hashes of particular data, this work is connected with a specific digital object. With a partial hash collision system, you have a device that can demand a precise quantity of computing work—a number of guesses—that anyone can verify as having been done, based on data you specify: hashcash.

BIT GOLD

Postage stamps, like mobile phone minutes, easily became currency. In the United States around the time of the Civil War, for instance, the dearth of small change led to a formal order for postmasters to no longer honor stamps that were "soiled or defaced,"

to prevent their monetary circulation.[13] "They would have just as much value, and would answer precisely the same purpose, so long as the community chose to take them," said the 1862 *New York Times* of the stamps, as they might of hashcash strings a hundred and thirty years later. But platforms would be needed to circulate these chunks of proof of work, "P.O.W. tokens," so that they could be reused—rather than the one-and-done of the hashcash postage for sending mail—and function not as metered postage but as something closer to money.

Hal Finney jumped into this expansion of Back's idea. He designed a system where a hashcash token could be sent to a special server, which would return a *reusable* proof of work (RPOW) token. You could spend this, redeem it, or otherwise transact it with someone, who would send it to the server in turn for another such token. "In this way," Finney wrote, "a single POW token is the foundation for a chain of RPOW tokens. The effect is the same as if the POW token could be handed from person to person and retain its value at each step," like cash. It would—at least in theory—retain its value, and the one-time transactions meant you could not copy-paste the same chunk of difficult hashcash to spend it repeatedly. It would rely on a "transparent server" system he was developing: a way for everyone to verify that the proof-of-work renewal system was working properly—neither duplicating nor deleting—without making the server itself vulnerable.

Finney sketched out applications for this strange vehicle of work and value. He described a kind of poker with RPOW tokens functioning as chips, and envisioned a version of the peer-to-peer file-sharing protocol BitTorrent that rewarded people with RPOW tokens for making their downloaded files available to others, and the tokens could in turn be used to pay for a faster spot in the download queue next time—a bit like the

"CryptoCredits" Tim May proposed as the internal currency of BlackNet. With such a device, in other words, you could build systems akin to metered postage, credit card reward points, and a casino's system for redeeming chips.

Nick Szabo discussed how proofs of work might function as something more akin to gold, a scarce commodity. Szabo had worked with David Chaum on the digital cash system at Eindhoven and corresponded on the cypherpunk list. (He appears among the Extropians in the next chapter, and is one of several people proposed as the identity behind the Satoshi Nakamoto pseudonym as the creator of Bitcoin.) In the late 1990s, in conversation with Finney and others, he toyed with the idea of using a hashcash-style technology to create a store of value he called *bit gold*. In a 2001 paper, Szabo referred to Finney's RPOW as an implementation of "a version of bit gold" (and he thanks Mark Miller, the Xanadu programmer, for his comments and encouragement).[14] "Unforgeably costly bits," he argued, "could be created online with minimal dependence on trusted third parties, and then securely stored, transferred, and assayed with similar minimal trust."[15] The costly bits would be the result of a proof-of-work computation on a set input—the "challenge string"—which is derived from the most recent verified bit gold proof of work, linking them together.

The new bit gold proof of work would be time-stamped and signed into another system of Szabo's, a "distributed property title registry"—an "unforgeable . . . chain of digital signatures" granting control over pieces of bit gold to their owners. As they were sold and exchanged—their ownership signed for and reassigned—these proofs, unique and variably valuable, would be grouped together into useful chunks, akin "to what many commodity dealers do today." It is a mechanism that prefigures aspects of the Bitcoin blockchain, a distributed ledger whose

"coins" consist of nothing but ownership assigned through proof of work and a chain of digital signatures.

Something like a casino, something like a postal system, something like the gold desk at a commodities broker. What if you could take this technology a step further: to build something like a bank—with a kind of money that was not just on the network's hardware, but *of* it? What would a hashcash-based bank look like?

THE IMPOSSIBILITY OF VIOLENCE

The standard mental model of the secret cypherpunk bank—a model inherited from physical banks—went like this: There is a central server, a computer that stores a list of accounts and the amounts of some kind of "digital coin" assigned to each account. I have five coins, and you have ten. In exchange for services rendered, you log in to the bank and send three of your coins to my account. The "coins," some string of letters and numbers, never leave the bank's server; they get reassigned to one or another account. These "coins" may be issued with reference to grains of physical gold stored in a safe-deposit box—a lemonade-stand version of the New York Federal Reserve's gold storage, where ownership transfers involve moving marked bars between compartments or shelves in compartments without ever leaving the facility. The bank is "encrypted" in the sense that our transactions and accounts are anonymized, and the computer on which all this activity takes place is likewise encrypted.

This presents two problems.

First, what happens if the server goes away, temporarily or permanently? The server on which the bank lives must physically sit somewhere: in the closet of some Gadsden-flag-flying libertarian who's already on a police list for their cannabis legalization activities, or in an office park that's just lost its lease,

or a hurricane-prone Caribbean jurisdiction. Can it be defended?

Second, and worse, how sure are you about the person in charge of the server, which is also the bank? How are you sure that the digital "coins" in your account at the covert bank are really backed by anything? Can the administrator make and sell as many as they want? Is that gold ingot on the digital scale next to the day's newspaper in the photograph real or not? Is their security absolutely airtight, or could hackers empty out accounts or make copies of the same "coin" and spend it as often as they like?

How to secure the virtual bank? Sustaining the fantasy of intangible, untouchable cyberspace required the metaphorical "'solidity,'" May wrote, of "walls, doors, permanent structures" provided by encryption—but as a practical matter that other space was defined by a constantly leaking permeable membrane.[16] Signals pass through walls, people keep passwords and addresses on sticky notes for reference, and computers and servers and digital media are physically seized, bagged, and put in the vans of Interpol or the FBI. Someone with a powerful electromagnet—or just access to the fuse box—could damage or destroy a covert messageboard or secret bank, for any reason, from deleting accumulated debts to making mayhem for its own sake (never an impulse in short supply on the Internet). "Physical security is needed," May argued; you could not—yet—really run a network wholly apart from the planet on which it was embedded.[17]

Some of the machines involved needed "controlled access" and protection—one of the longer-term strategies was to run crypto anarchist networks on satellites, which would be much more difficult to shoot down. Some of the cypherpunks anticipated potential reprisals or dangers from extortionists or

criminal cartels. The work of securing the network was not just a matter of mathematics after all, but required hardware, facilities, and tradecraft. "For much the same reason no 'digital coin' exists," May concluded: you couldn't rely on the security of the machinery to secure the transaction records and the mint. Without control of a built environment, the systems of exchange and accumulation would never escape the suspicion of vulnerability. The Other Plane, disembodied cyberspace, needed physical spaces. It needed zones, sovereign territories, spaces of exception—a trajectory that would one day lead to an abandoned artillery platform in the North Sea.

Tim May and Ryan Lackey—a technologist who would later turn up on that North Sea platform, hoping to build an anonymous bank—argued the merits of scopes on Dragunov guns and the relative utility of the AR-15 assault rifle, generally in the context of fighting a government raid. Armed resistance against such a physical assertion of state sovereignty was, they conceded, a terrible idea. "I figure that if I'm ever in a situation where I have to engage multiple targets quickly," Tim May wrote in response to a detailed analysis of "Soviet-style weapons" in raid defense, "I'm probably a goner."[18]

"I don't understand why there is so much talk about guns here lately," Dai wrote on the mailing list in January 1998. "Unless someone comes up with a weapon that has some very unusual economic properties, individuals cannot hope to compete with governments in the domain of deadly force."[19] (Economic weapons: One thinks of the rays developed by the scientific vigilantes in Technocratic sci-fi, which blank banknotes and turn gold into tin to soften the human terrain for their rational coup.) "Think about it," he continued: "if we can defend ourselves with guns, why would we need crypto?"

On November 27, 1998, Wei Dai posted a proposal to the cypherpunk mailing list for "a new protocol for monetary exchange and contract enforcement for pseudonyms," which he called "b-money."[20] The idea would, in retrospect, loom large, but he mentioned it in passing. The link appeared at the end of a note about PipeNet, a project to shuffle messages using encrypted communications on the network to make it difficult for an adversary to figure out who is speaking to whom. (It was akin to the Onion Routing system that became Tor.) His b-money text file began, "I am fascinated by Tim May's crypto-anarchy. . . . It's a community where the threat of violence is impotent because violence is impossible, and violence is impossible because its participants cannot be linked to their true names or physical locations."

What Dai was proposing was something different, though, from May's envisioned "digital coin" that needed physical protection for the machinery, like a bank. It was a form of cash that was built on the very mechanisms it also used for transactions. It was a currency that was itself wholly *cryptographic* in its mechanisms, and not just encrypted in its transactions. It was the first mint that belonged natively to the permanent frontier. "It's almost too simple to describe," wrote Hal Finney about b-money: "In principle, it is just a matter of everyone keeping track of how much money everyone else has."[21] That's part one, and brings us back to the second problem with the fantasy of the cypherpunk bank: How do you know you can trust the banker?

MONEY IS NOT ABOUT ATOMS

Dai's solution was to deconstruct the bank into distributed components: a set of accounts that can hold money, a mechanism for transacting the money between accounts, and a means of

issuing that money in the first place. Then, rather than the bank being some central location (literally and metaphorically) to which all its clients would refer, every client working together would constitute the bank collectively. Dai exploded "the bank" outward, into a decentralized network composed of all of its participants.

All the accounts on Dai's network are pseudonyms, in the now-familiar public-private key arrangement, and each pseudonym keeps a copy of the ledger of the whole bank: "everyone keeping track of how much money everyone else has." Finney continued his description: "Whenever there is a transfer of money, this fact gets broadcast and everyone updates their databases."[22] When you spend some money, that act is announced to the whole network (signed with your private key); everyone checks their ledgers, and, if you have the money to spend, they update accordingly: you debit three, I credit three.

Finally, and most ingeniously, anyone on the network can produce new money according to a set of collectively agreed-upon rules. In the case of Dai's first version, new money can be added to the system by broadcasting "the solution to a previously unsolved computational problem." It must be easy to determine that the solution is true, and likewise to measure exactly how hard the problem was to solve, so that the difficulty can be calibrated in terms of "a standard basket of commodities." Minting would therefore be challenging and moderately expensive, but not impossible, and pegged to the price of some mix of barrels of oil, bushels of grain, feet of lumber: as money became scarce and more valuable, it would be worthwhile to expend computational work on minting more of it; as more was produced, the supply inflating and dropping in value, fewer people would spend the computing energy—the work, the

money—to mint, and the money would become more valuable again.[23]

The heart of the project lay in rebuilding money on computational, and specifically cryptographic, lines: public and private keys for identity and authentication, untraceable networks for transaction, and some well-established way of setting computational problems to be solved. (Dai also discussed how the same tools used to reconcile transactions could be used to set and validate contracts.) For the problem-setting, as Back pointed out, partial hash collision systems were an ideal fit; he gave public feedback within days to Dai's original proposal. Abstracted from an institution or an established group of people, b-money was *of* the network rather than on it, built of cryptographic tools rather than simply concealed by them, native to the Other Plane.

A decade later, Satoshi Nakamoto, Bitcoin's pseudonymous creator, wrote to Dai: "I was very interested to read your b-money page. I'm getting ready to release a paper that expands on your ideas into a complete working system"—the "complete working system" that would be Bitcoin.[24] "Adam Back (hashcash.org) noticed the similarities and pointed me to your site." In January of the next year, Nakamoto followed up: "I just released the full implementation of the paper I sent you a few months ago, Bitcoin v0.1. . . . I think it achieves nearly all the goals you set out to solve in your b-money paper."[25]

In 2002, Finney ended his summary of b-money by putting the matter into a larger context: "The important point in these conceptions of money is that it is fundamentally a form of information. B-money shows that most clearly. Money is not about atoms, it is about bits. Extropians should shun old-fashioned views of money as based on material goods."[26]