MARCH 2018

VOLUME 33 NUMBER 3

Blockchain - Blockchain Fundamentals

By [Jonathan Waldman](https://msdn.microsoft.com/en-us/magazine/mt149362?author=jonathan+waldman) | March 2018

Back in 1999, the file-sharing network Napster made it easy to share audio files (usually containing music) on a hybrid peer-to-peer network (“hybrid” because it used a central directory server). That file-sharing network did more than just share music files: It allowed all users to retain copies of those shared files such that a single digital asset resulted in a limitless number of perfect copies across a global network. The casual ease with which technology was leveraged by anyone with a computer caught venerable Tower Records by such surprise that it was forced to close all of its 89 U.S.-based stores by 2006.

In 2008, the subprime debacle occurred, during which long-­established, powerful U.S. financial institutions and insurance companies declared or teetered on the brink of bankruptcy. Those circumstances called for immediate federal government intervention in order to avoid a domestic and possibly global financial meltdown. This important event left a populace leery of centralized banks and exposed the danger of having financial ledgers closed to public scrutiny. In March 2008, the Heartland Payment Systems data breach exposed more than 130 million credit card numbers, many of which were later used to make fraudulent purchases.

These events illustrate the perils of living in a digital, connected world that depends on transaction-fee-generating middlemen and leaves people vulnerable to digital exploits, greed and crime. The academic challenge became how to create an available, disintermediated digital infrastructure on which a digital asset can be openly and reliably transferred (rather than copied and shared) from owner to owner, that has no corruptible or fallible central authority, is secure, and can be trusted.

**Enter the Bitcoin Blockchain**

Seemingly in response to the historical context in which it happened, on Jan. 3, 2009, a new kind of infrastructure mined 50 digital coins and recorded them on a tamper-proof public ledger that’s replicated on a decentralized peer-to-peer network of Internet-connected computers. Those 50 cryptocurrency units, called bitcoins, were recorded as the genesis block—the first link on what would come to be known as the Bitcoin blockchain.

The remarkable thing about this blockchain-powered cryptocurrency is that it lacks any sort of trust authority or governance (such as a bank or government) to validate each transaction. Furthermore, it disintermediates transactions, making it possible to transfer digital currency internationally using a global network with no middleman involvement, such as a broker or agency. Due to its reliance on modern cryptography, the data contained in the blockchain is tamper-proof and pseudonymous. And because a given blockchain is replicated on every node that comprises its peer-to-peer network, there’s no single point of failure, making the technology available and reliable.

Since Bitcoin’s launch, blockchain technologies continue to mature rapidly. Their implementation details vary dramatically, making the study of blockchains dynamic, vast and complex. Indeed, the term “blockchain” no longer applies only to Bitcoin in particular or to cryptocurrencies in general. Blockchains are being refined and perfected so they work faster and smarter. In fact, some blockchain technologies allow scripting to support smart contracts, which permits custom rules to be applied to transactions. In this way, blockchains have morphed into a new kind of programmable, hacker-proof storage technology, which is one reason why IT professionals, businesses, financial institutions, and others are clamoring to harness their true potential.

If you’ve been sitting on the blockchain sidelines, it’s time to catch up. In this introductory article, it isn’t possible to cover the finer technical details of any particular blockchain technology—each has its own rules, features and customizations—but the ideas I discuss here will help orient you to the core technical underpinnings on which many modern blockchain technologies are based.

**How Blockchains Work**

The Bitcoin blockchain is the world’s first practical example of blockchain technology. And because of that distinction, “blockchain” is often misunderstood as being synonymous with Bitcoin. However, modern blockchain technology offerings track digital assets other than a digital currency, and those blockchains work quite differently from the way Bitcoin’s blockchain works. Additionally, the Bitcoin blockchain popularized the notion that a blockchain is a data structure that virtualizes a bank ledger by tracking credits and debits while offering a creative, cryptographic solution that effectively bars the double-spending of cryptocurrency units. For this reason, the terms “digital ledger” and “double spend” became associated with cryptocurrency blockchains. Yet these terms, respectively, broadly apply to tracking ownership and enforcing a single transfer of digital assets. When you see these terms, don’t assume they refer solely to cryptocurrency-oriented blockchain technologies.

At its essence, a blockchain is a tamper-proof data structure that tracks something of value or interest as it passes from owner to owner. That “something” can be any kind of digital asset—such as a digital coin, a Word document or the serial number of a Microsoft Surface tablet. In fact, any item that can be associated with a unique digital fingerprint can be tracked on a blockchain. Blockchains solve the so-called “double-spend” problem by requiring that ownership of a digital asset be transferred rather than copied or shared. But what makes a blockchain technology interesting is that it establishes a protocol, enforces transaction rules, and is able to let the nodes on its distributed network of computers self-police the entire operation. And it accomplishes this remarkable feat with no central server or trust authority, speedily and globally (that is, internationally). This promise excites those who see it as a way to eliminate middlemen and reduce or waive transaction fees, making commerce more efficient for businesses and consumers alike.

**Blockchain Core Components**

The Bitcoin blockchain network is public—anyone can participate anywhere in the world. Yet newer blockchain offerings, such as the Microsoft Azure-hosted blockchain, can be configured as public, private or permissioned networks. Blockchains are considered to be decentralized, but that term requires clarification: As Vitalik Buterin explains ([bit.ly/2tEUYyT](http://bit.ly/2tEUYyT)), “decentralized blockchains” means they’re “politically decentralized (no one controls them) and architecturally decentralized (no infrastructural central point of failure) but they are logically centralized (there is one commonly agreed state and the system behaves like a single computer).” Decentralization offers fault tolerance, attack resistance and collusion resistance (the meaning of this will become clear when I discuss proof-of-work later).

Understanding how to engineer a public blockchain requires knowledge of cryptographic hashes, public key cryptography (PKC), binary hash chains (Merkle trees, in particular), and consensus algorithms. I’ll briefly review these concepts, and then I’ll show that a blockchain is a hash chain that contains a hash chain of transactions. Once you grasp this nested-hash-chain concept, you’ll understand blockchain technology’s fundamental design.

**Cryptographic Hashes** While there are many one-way crypto­graphic hash algorithm variants, a popular choice is to leverage SHA-256 ([bit.ly/29kkpft](http://bit.ly/29kkpft)), a one-way hash function that accepts a message of up to (264-1)/8 bytes and returns a 32-byte hash value (64 hexadecimal characters) in the decimal range of 0 to roughly 1.16 x 1077. To put that number’s magnitude in perspective, a drop of water has about 5 x 1012atoms; the observable universe is estimated to have in the range of 1078 to 1082 atoms. Tweaking any character in the message and re-computing the SHA-256 hash value generates an entirely new hash value. (To experiment, visit [onlinemd5.com](http://onlinemd5.com/) and set the file or text checksum type to SHA-256.)

Given the same input, the SHA-256 algorithm always produces the same fixed-length output. With respect to blockchain technologies, the value of using SHA-256 cryptographic hashes is that they’re unique enough to serve as a sort of digital fingerprint while also acting as a checksum. Furthermore, one-way hash functions can’t (as a matter of practice) be decoded. Consider the SHA-256 value for my name: 8F12D83BA54AC0EA7687AD4AFDE5E258BBFF970AA8D60C6588381784C502CA8E. Given that hash value, there’s no practical way to algorithmically reverse it back to my name. (One hacking technique leverages rainbow tables that list already-calculated hash values for common strings, such as “password”—but that’s not algorithmically reversing the hash. To thwart such exploits, it’s customary to embellish the string to be hashed by tacking on a random string, known as a “salt” value.)

If you don’t have a SHA-256 generator handy, consider the table in **Figure 1**, which shows how strings of various lengths always produce a 64-digit hexadecimal hash value, and that a small change to any particular string produces a completely different result.

**Figure 1 Hashing Strings of Various Lengths Using the SHA-256 Algorithm**

|  |  |
| --- | --- |
| Input String | SHA-256 Hash Value |
| m | 62C66A7A5DD70C3146618063C344E531E6D4B59E379808443CE962B3ABD63C5A |
| M | 08F271887CE94707DA822D5263BAE19D5519CB3614E0DAEDC4C7CE5DAB7473F1 |
| M1 | 2D214CA69B86C255BE416D42CCA977A59B34A7492873105522C35015FAB806F0 |
| M2 | 0892A10ECE1F933EE98F5D554601B28F8437801D1AA1B77799E4035DDCB6950C |
| March | 9D95A2CF0D7180B5089691163B188A7203B0CDE179346B8CFAA8AB6C2C3E6414 |
| March 1, 2018 | 767328E7367048FA9DB37354CFA43DBB1691E8330DB54D54F52C1A444CA2E680 |
| March 2, 2018 | CCF33BF1C08B74EDE6A7C15C56EEC16269D83967670032ACDA6EE395361B7595 |

Sometimes a hash value is double-hashed, which means that the first hash is hashed again by applying a second round of the SHA-256 algorithm. If I double-hash the values in **Figure 1**, I end up with the results in **Figure 2**.

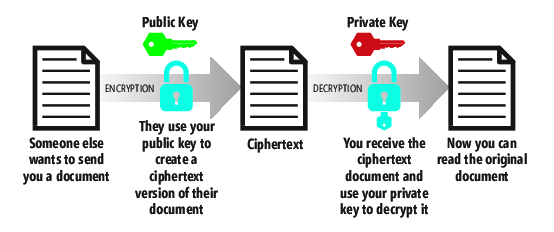
**Figure 2 Double-Hashing the Values in Figure 1**

|  |  |
| --- | --- |
| Input String | Double SHA-256 Hash Value |
| m | A5FCE7E78734EC317F83F9019C80FFAF2508689B06EFA02191495A7D21FECE9A |
| M | 6F6DCF58526B0D29EE664A708A939B7CDAC124A6A8569FCACE46FEAD38868E2E |
| M1 | 6C5D08BE9FFBBABD24B5F19AFFE6590FD402D347A50B519A59D40E15DCC0A6CB |
| M2 | B2307311CC5877D5A581EDC821F3BFD5F99EB4E3B1D1B4009D9545BCF07E2E1A |
| March | B5410E155022AE6EB22CA21FADEDE65F0F7296DE14CA1D7A720A4937BD23AA5D |
| March 1, 2018 | 345DD725FEE80F8C5953A66C1495605E4ED01C4CE5AEF6C0A6D238999266A1A6 |
| March 2, 2018 | 3E85B3D910BA77F88ECD5E24D1396457C532C73B89C032DED9AD0CBB4D4D9794 |

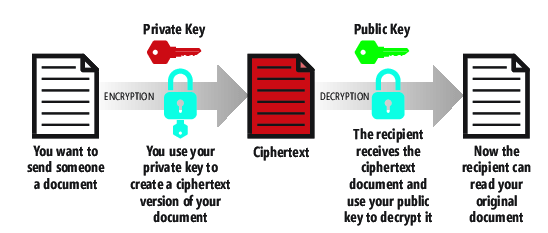
**Public Key Cryptography** Recall that one of the primary functions of a blockchain is to track ownership of a digital asset. The digital asset in question may be worth nothing or many millions of dollars, so the ownership test must ensure that the owner can’t be spoofed. To conduct such a test in the digital realm, blockchains leverage PKC, which enables the owner to digitally sign their asset in order to prove ownership and authorize it to be transferred. Unlike symmetric key encryption, wherein a single private (secret) key is used to both encrypt and then decrypt a message, PKC uses asymmetric key encryption.

Because an accurate validation algorithm of digital asset ownership is critical for blockchains, they employ a high-strength public/private key-pair generation strategy that relies on the Elliptic Curve Digital Signature Algorithm, or ECDSA. The beauty of ECDSA is that it creates keys that are shorter in length but cryptographically stronger than same-length keys generated by the usual algorithm: Digital Signature Authorization (DSA). Whenever they’re needed, users access a software application that uses ECDSA to generate the cryptographic key pair. The user must retain a backup of the private key because that key is required to transfer or harness the value held in a digital asset that’s stored on a blockchain. If you have access only to the private key in a private/public key pair, you can regenerate the public key because there’s a mathematical relationship between the two keys. But the private key can’t be generated from the public key (which means if you back up only one key, be sure it’s the private key!).

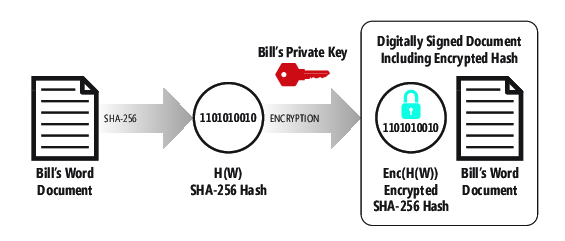
These keys typically are used in one of two ways. The first use case (see **Figure 3**) is when you want someone to send you an encrypted message that only you can open. To do that, give the other person your public key and ask them to use it to encrypt the document using software that applies an encryption algorithm and produces a ciphertext—the encrypted version of their message. They then send you only the ciphertext. Because they used your public key to encrypt the document, you must use the correctly paired private key to decrypt it.

   
**Figure 3 Using PKC When You Want Someone to Send You an Encrypted Document/Message That Only You Can Open**

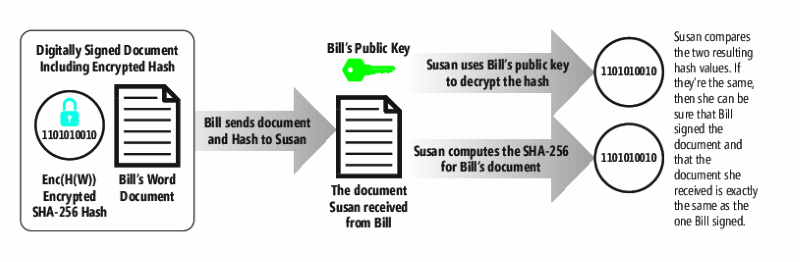
The second use case (see **Figure 4**) is when you want to encrypt a message and demonstrate that it is indeed from you. To do that, you use your private key to create a ciphertext of your document. You then send that ciphertext to someone else. They use your public key to decrypt it. Because only your public key can decrypt that document, the recipient can assume that the document was encrypted by your private key—and unless your private key has been misappropriated, the document came from you.

   
**Figure 4 Using PKC When You Want to Send Someone an Encrypted Document/Message to Assure Them That It Indeed Came from You**

A third use case employs PKC to prove ownership of a digital asset through a digital-signing process. In this use case (see **Figure 5**), imagine that Bill has composed a Word document of a legally binding document that he needs to e-mail to Susan. Susan wants to be sure that the copy of the document she receives from Bill is actually from Bill and hasn’t been tampered with en route. Bill first creates a SHA-256 hash of the Word document and records the value as H(W). He next uses his private key to encrypt the document hash, resulting in Enc(H(W)), and then sends the Word document (optionally encrypted) and the Enc(H(W)) value (this is Bill’s digital signature for document W) to Susan.

   
**Figure 5 Using PKC Along with a Cryptographic Hash to Digitally Sign a Document/Message**

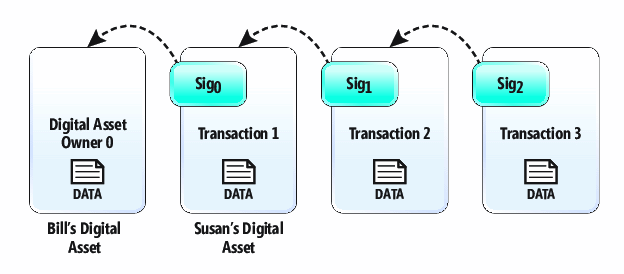
Susan re-computes the H(W) value from the copy of the Word document she received, then uses Bill’s public key to decrypt the Enc(H(W)) value (see **Figure 6**). If the hash Susan computed is equal to the decrypted H(W) value, Susan can conclude that Bill signed the document and that the copy she received is exactly the same as the one Bill signed.

   
**Figure 6 Using PKC Along with a Cryptographic Hash to Verify That a Document/Message was Signed by the Expected Party**

Using hashing and PKC, a blockchain maintains a history of digital asset ownership using transactions. Transaction data objects are linked to each other, forming a data structure called a hash chain. The way this works is that each transaction record constitutes the message (m) that’s hashed using function (H) and then signed using the owner’s private key (s). (It’s customary to use “s,” for “secret,” to represent the private key so as not to confuse it with “p” for the public key.) This produces a signature (sig):

sig = signature(H(m), s)

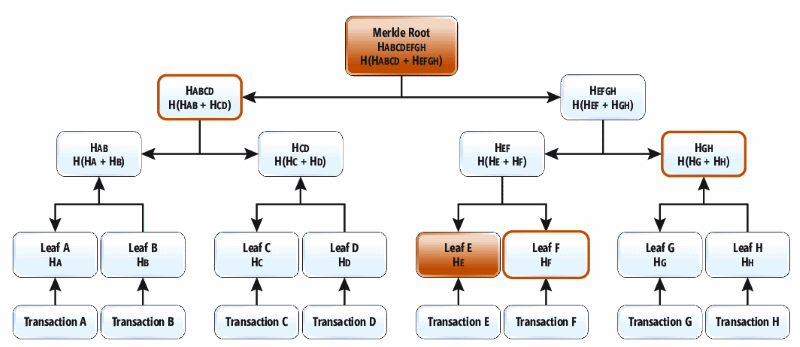
When a digital asset transfers from one owner to another, its digital signature is examined, verified, and digitally signed by the new owner, and then registered as a new node on the hash chain. Although the details of the implementation vary dramatically across blockchain technologies and versions, the basic idea is the same for all of them. For example, as shown in **Figure 7**, Bill is the owner of a digital asset and uses his private key to initiate a transfer of that digital asset to Susan. Susan’s transaction record uses Bill’s public key to verify his signature. After this, Susan’s public key is used to sign the digital asset, making Susan the new owner. This creates a new transaction record—a new link on the transaction hash chain.

  
**Figure 7 The Transaction Hash Chain Uses Digital Signatures to Transfer Ownership of a Digital Asset; Each Transaction Record Maintains a Cryptographic Backlink to the Previous Transaction on the Hash Chain**

This hash chain of transactions is cryptographically secure and tamper-proof. Any change to Transaction 0 would cause Sig0 to change, and that would require an update to the hash value stored in Transaction 1 and every subsequent transaction on the hash chain.

The transaction objects here are pictured with data. The data contained by each transaction varies for each blockchain implementation, so I’ve abstracted the underlying data for our purposes because the main point to understand is that the hash chain is a cryptographically linked chain of transactions—linked by the hash value of the previous owner’s transaction record. (In crypto­currency blockchains, each transaction object contains a list of digital-currency inputs and outputs, along with metadata, such as a timestamp and optional transaction fee. These cryptocurrency inputs and outputs provide the transaction detail needed to accurately model a financial ledger.)

**Merkle Trees** Some blockchains bundle up transactions using another kind of hash chain: the binary hash chain, or Merkle tree. A complete Merkle tree is referred to as a binary tree structure because it branches twice at each level starting at the root, as shown in **Figure 8**.

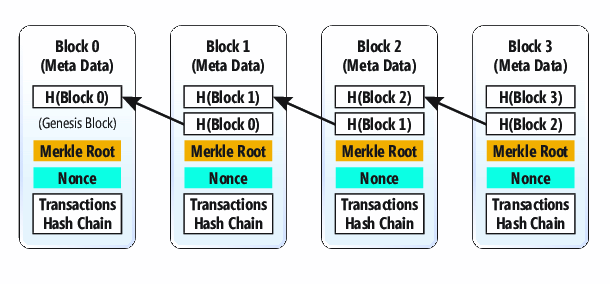
  
**Figure 8 A Merkle Tree is a Type of Binary Hash Tree That Produces a Merkle Root Hash; This Data Structure Can Efficiently Add Leaf Nodes and Calculate a New Merkle Root Without Requiring Full Re-Computation**

The work in setting up a Merkle tree is to create a series of leaf nodes by computing the SHA-256 hash for the data contained in each transaction object (the Bitcoin blockchain double-hashes each Merkle node; double-hashing can help strengthen the cryptographic value in the hash result should a vulnerability be discovered in the SHA-256 algorithm). The Merkle tree requires an even number of leaf nodes—it’s customary to duplicate the last leaf node if starting with an odd number. Then each pair of leaf nodes is hashed together, producing a new hash value. In **Figure 8**, Leaf A shows the hash for Transaction A as HA; Leaf B shows the hash for Transaction B as HB and so on. This pattern continues at each tree level until you reach the final root node. The root node’s hash value is the crypto­graphic hash sum of all of the other hash sums in the tree. Any change to the data in any of the leaf nodes causes the recomputed Merkle tree root hash value to change.

The Merkle binary hash tree structure offers some advantages. For example, it makes it easy to update data within a transaction and compute a new Merkle root hash without having to build the entire Merkle tree from scratch. For example, if Transaction E changes (it’s highlighted in **Figure 8**), all you need to do is walk the tree efficiently back to the Merkle root, computing new hashes once for each level. Thus, you first compute the new Leaf hash HE; then compute HEF from HE and HF; then compute HEFGH from HEFand HGH; then compute a new Merkle root hash from HABCD and HEFGH. Updating the Merkle root hash required only four computations versus the 15 required to build the Merkle tree from scratch!

**Building a Blockchain**

To build a blockchain (see **Figure 9**), the binary hash chain data object containing transactions must somehow be committed to a tamper-proof data store that everyone can use (remember, this is a public blockchain—any node on the network can read from or write to it). The Merkle tree structure contains transactions and is tamper-proof, so it would seem it could serve as the blockchain. But there are several problems. In order for Bill to send his digital asset to Susan, Bill must trust the service or Web site that acts as an agent to process his digital-asset transfer request, and he must trust the server that persists the hash structure. Without a central node to process a new transaction or a central authority to delegate them for processing, any node could process Bill’s pending transaction. A rogue or dominant node having superior processing power could allow invalid or fraudulent transactions to occur and those could propagate to honest nodes. To solve that, the network could try to randomly assign a node to process Bill’s transaction, but that again centralizes control and requires trust that the random number generator is indeed enforcing randomness. To eliminate this issue, blockchains use consensus algorithms, described next.

  
**Figure 9 The Blockchain Is Composed of Blocks That, in Turn, Include Transaction Hash Trees; Blocks on the Blockchain Are Back-Linked to Previous Blocks and Are Validated Using a Proof-of-Work Algorithm**

**Consensus Algorithms** Blockchain technologies avoid centralized data store and trust-authority issues by honoring a protocol that dictates how blocks are added and maintained. They do so by enforcing a blockchain-building consensus algorithm. There are different kinds of consensus algorithms—I’ll discuss how the proof-of-work (PoW) algorithm works.

PoW is based on the idea that a node on the network needs to prove its honorable intentions by incurring a cost and consuming the time required to solve a computationally challenging problem. To induce a node to participate in such a system and play by its rules, the network offers an incentive—often money—that is, node operators get paid when they add a block to the blockchain. To earn that money, nodes must validate all transactions (to ensure that they meet a blockchain’s specific rules) and then solve a cryptographic puzzle.

Earlier, I mentioned that a central authority could randomly assign a node to process a batch of new transactions. That approach would require a central random-number generator, which could be flawed, hacked or disabled. However, giving nodes a difficult puzzle to solve yields the desired effect: The node that will first solve the puzzle can’t be determined in advance, making for a sort of node-lottery on the network. No central authority is needed—and that’s one of the key innovations of blockchain technologies. I also mentioned that blockchains are decentralized and therefore offer “collusion resistance.” Because PoW takes time and costs money in computational power, it’s practically impossible for any single node or group of nodes to collude on the network and have a blockchain-making advantage over other peer nodes. (There is a “51 percent attack” risk that suggests collusion can occur if a group of nodes ends up with 51 percent of the computational power, but employing a PoW consensus algorithm makes the possibility of such an attack infeasible.)

To construct a block of transactions, a node grabs unprocessed transactions stored on the network and builds a Merkle tree in order to compute the Merkle root hash. Therefore, a candidate block contains a list of transactions along with a block header that includes the Merkle root hash value, a current timestamp and PoW difficulty level (and sometimes additional header data). Then it must solve the PoW puzzle, which involves computing a double hash of the entire 256-bit block hash value, then finding a 32-bit number, called the nonce, which can be concatenated to it so that the hash of the resulting 288-bit number produces a result that has a certain number of leading zeroes. The 32-bit nonce has a range of 0 to 232 (4,294,967,295), so instead of just trying to guess the nonce, it’s typical to start with nonce 0, generate the SHA-256 hash, and determine if it has the target number of leading zeroes (that is, the resulting hash is below a target value); if it doesn’t, the node increments the nonce value and tries again. If the node attempts all nonce values without solving the puzzle, its recalculates the block hash value. This guarantees the production of a different block hash value because a timestamp in the block header is included in the block-hash calculation. At any time, a node can select a different batch of pending transactions for inclusion in the new block (or add new pending transactions that might have appeared since it last checked), and this changes the Merkle root hash value, which, along with the timestamp, changes the newly computed block hash value. Each time the block hash is recomputed, the node iterates over all 4 billion-plus nonces again.

In time, some node on the network will solve its cryptographic puzzle. When it does so, it adds the new block to the end of its copy of the blockchain (every node maintains a copy of the blockchain) and then broadcasts the new block to every other node on the network so that they can update their blockchain copy. When a node broadcasts a new block, other nodes don’t simply trust that a new block is valid—they prove it to themselves by validating the block. To validate, a node simply verifies the PoW puzzle solution by computing the block’s SHA-256 hash concatenated with the nonce value and verifies that the answer produces a hash that has the number of leading zeroes specified by that block’s PoW difficulty value.

Incidentally, on some blockchains, the PoW difficulty value is continually adjusted by the protocol so that new blocks are added to the blockchain at a prescribed time interval. This continual adjustment is necessary because nodes are constantly appearing and disappearing from the network and thus the average computational power of the nodes is always changing. Remember that in PoW, there is an incentive to add blocks to the blockchain, so node administrators often beef up their hardware in order to compete for an award. On the Bitcoin blockchain, the difficulty value is tweaked every 2016 blocks so that blocks continue to be added at an average rate of 10 minutes per block.

Sometimes branches occur. That’s because on a large network, new-block propagation takes time. It’s possible that during propagation, another node solves its PoW puzzle, adds a new block to its copy of the blockchain, then broadcasts that blockchain on the network. Receiving nodes will always add a valid block to their copy of the blockchain—and because each block is cryptographically tethered to the previous block, two new blocks published by two different nodes will result in a branch linked to the same block at the end of the chain. That’s OK, however. Over time, nodes will add new blocks to the end of what the protocol deems to be the “longest chain.” For example, given a branch, the longest chain could be defined as the one with the most-recent block timestamp. Over time, a single branch will prevail in length and blocks from abandoned (shorter) branches will be removed, returning their transactions to the pool of unprocessed transactions.

**Wrapping Up**

In this article, I’ve shown how to construct a public blockchain that’s composed of cryptographically linked blocks—each of which contains its own hash chain of cryptographically linked transactions—on a decentralized peer-to-peer network of nodes. I covered the basics of blockchain technologies, trying not to focus on any single implementation but instead concentrating on some of the more typical technical features they all share. If you want to explore the subject further, I suggest choosing a specific blockchain technology, such as Bitcoin, Ethereum or Ripple, and trying to master the details of its particular implementation. If you want to experiment with blockchains on your own, take a look at Azure-hosted blockchain offerings at [bit.ly/2Gj2zaC](http://bit.ly/2Gj2zaC).

**Jonathan Waldman** *is a Microsoft Certified Professional who has worked with Microsoft technologies since their inception and who specializes in software ergonomics. Waldman is a member of the Pluralsight technical team and he currently leads institutional and private-sector software-development projects. He can be reached at*[*jonathan.waldman@live.com*](mailto:jonathan.waldman@live.com)*.*

 Thanks to the following Microsoft technical expert for reviewing this article: James McCaffrey