Hash Functions and Cryptocurrencies

Erik J. Boswell

October 2021

Outline

- Cryptocurrencies
- Hash Functions
- Properties of Hash Functions
- SHA-256 Standard

What is a cryptocurrency?

A cryptocurrency is a non-trust based system of payment.

This means there's no trusted third party, like a bank, to handle payments for individuals, and there is no one central group controlling the actions of this cryptocurrency, rather the collective group using the currency is responsible for it's control.

While a cryptocurrency is usually decentralized, some cryptocurrencies are considered centralized when they are minted before issuance or when a single issuer is issuing the currency.

Bitcoin

The most classic example of a cryptocurrency is Bitcoin. Originating in 2009, this cryptocurrency has grown to be $50000-60000 in value for a coin now. Bitcoin is a classic example of a decentralized cryptocurrency as detailed in the original Bitcoin Paper released by Satoshi in October 2008. This paper sets the tone for a lot of what cryptocurrencies are, including terminology and development methods.

Initial Definitions for Cryptocurrencies

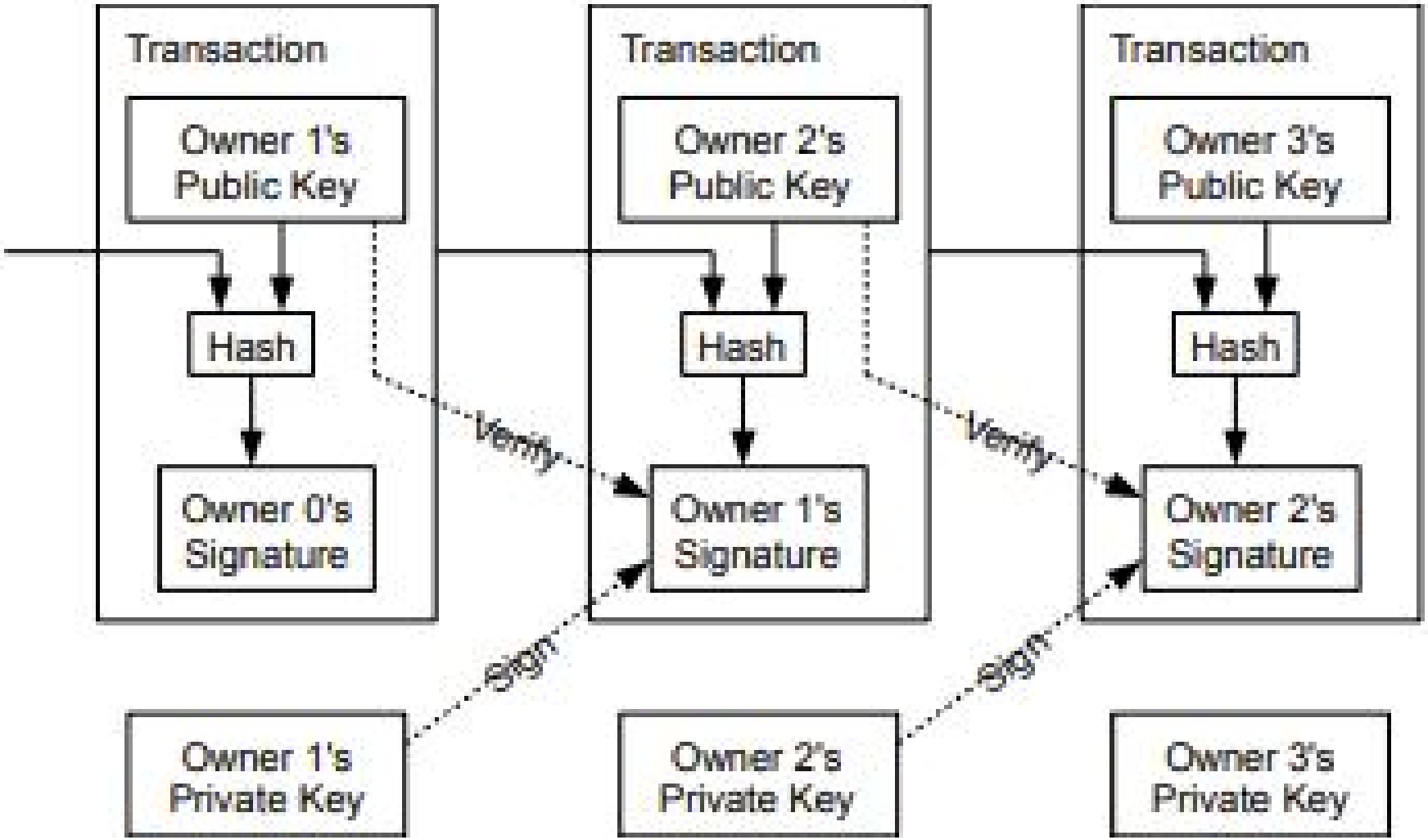
While one can possess cryptocurrency, what really makes up the cryptocurrency, and how much a person holds, is the ledger for said cryptocurrency. This ledger is formed by a block chain where each block stores a record of transactions verified with hash codes, and each block is chained together with matching hash codes.

Transactions

We begin with transactions for cryptocurrencies. We can start by thinking of these as regular transactions, whether they be electronic or cash based ones. The idea is that two people are trading value. With a cryptocurrency, there are several important elements in a transaction as follows

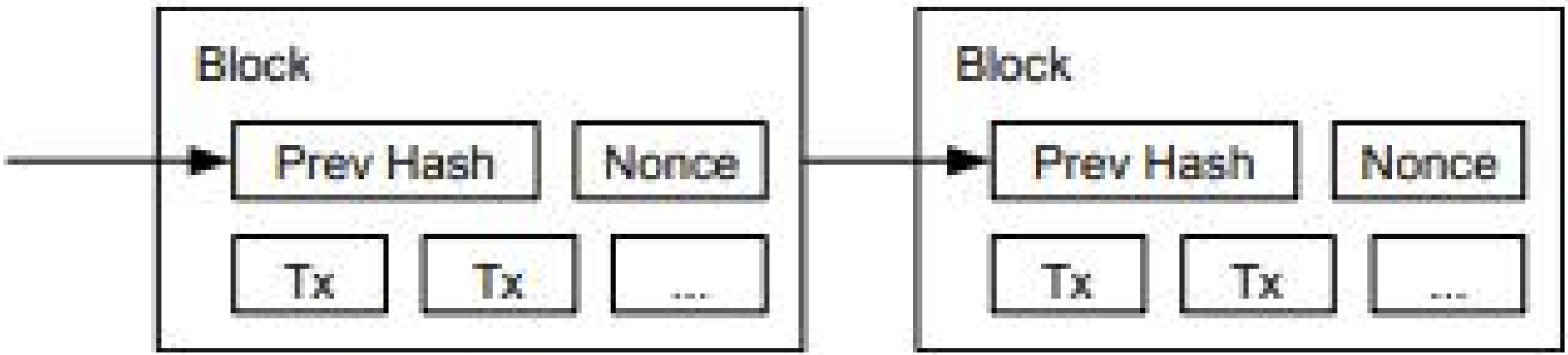
- Transaction ID
- Sender
- Receiver
- Amount of currency
- Signature

Transactions



The Blockchain

The block chain is a collection of blocks that point to one another with header and footer details in the block, and use hash codes to verify that blocks match each other.



Adding a Block to the Chain

When it comes time to add a block to the chain, there are several factors to be searched for.

- Previous Block Hash
- Completeness of Transactions
- Proof of Work

Hash Cryptography

Now we get to the core of the math behind cryptocurrencies. You've heard the words hash, hash code, and hash function used throughout this crash course explanation of cryptocurrencies. These all relate to hash cryptography, a one way encryption method based on taking strings like a transaction, translating them to binary and then passing them through a function to obtain a different binary value of a fixed length. These hash values are obtained by processing your initial binary value through a hash function to generate a hash code or hash digest.

What is a Hash Function

A hash function is a function that takes a term of arbitrary length, and converts it to a binary term of specified length, which we will give as

$$h : M_l \longrightarrow M_n \tag{1}$$

where $l \in \mathbb{N}$ is our initial message length in bits and $n \in \mathbb{N}$ is our fixed message length in bits for the output. Some examples of the hash functions used today are

- MD5
- SHA-256
- SHA-512
- SHA-3

Properties of Hash Functions

- Preimage Resistance
- Double Preimage Resistance
- Collision Resistance

Preimage Resistance and Second Preimage Resistance

Preimage resistance is a property of a function where it is infeasible to compute preimages. To define this, we want it so that for $h(x)$, from our hash function h, we cannot feasibly find x.

The property of being second preimage resistance is making it infeasible to find x' such that for $x' \neq x$, $h(x) = h(x')$.

Collision Resistance

Similar to second preimage resistance, collision resistance is defined on the case of not easily finding x and x' such that for $x' \neq x$, $h(x) = h(x')$.

Note that the slight difference between the two is the freedom of choice in x' for second preimage resistance, where collision is based on free choice of x and x'.

Examples of Basic Hash Functions

- Binning
- Modular Arithmetic
- Midsquare Method

Binning

Another example of a basic hash function which can accomplish preimage resistance almost immediately is binning. Binning is the concept of taking a set of values and partitioning them into disjoint subsets (preferably of equal size) and then mapping each subset to a value. So we can define this on a map

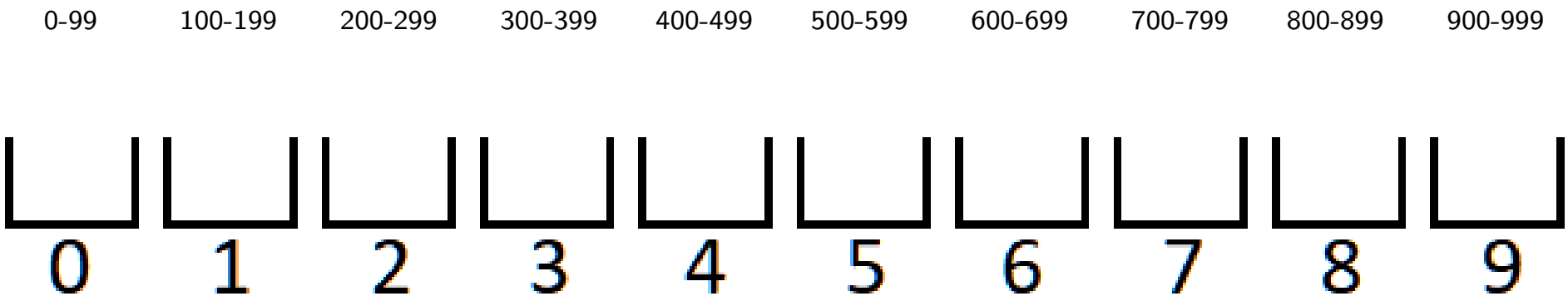
$$b : X \longrightarrow B$$

$$b(x) = i \in B$$

for $x \in X_i \subset X$. We make an example of the set $[0, 999]$ being binned based on the value of the third digit or division by 100.

Binning

As detailed before, if we bin the set $[0, 999]$ based on the hundreds digit, or division by 100 with no remainder, we have the following subsets of our initial set's partition and the corresponding values sent to each bin.



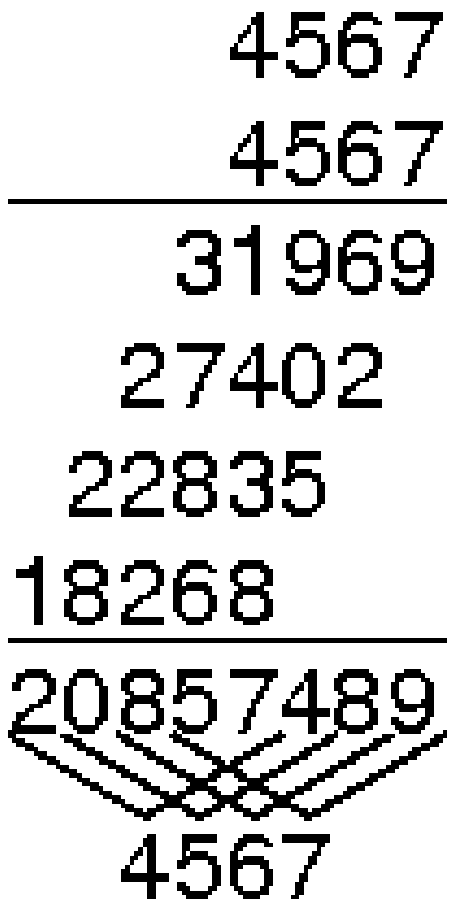
Modular Arithmetic

Modular arithmetic is based on working with elements modulo n an integer. This relation is defined by $a(\text{mod } n)$. Typically one reduces mod n, since this relation is defined by the equation $a = qn + r$, and $qn + r$ defines an equivalence class in the modulus, usually written as $[r]$.

- $3(\text{mod } 9)$
- $[3]=\{3,12,21,-6\}$
- $12 \equiv 21 \equiv 3(\text{mod } 9)$

Midsquare Method

The midsquare method follows squaring two integers and taking middle digits from the result. Suppose we have $x \in \mathbb{Z}^+$, then define $x^2 = d_1 d_2 d_3 ... d_n$ for n digits. Then we take r digits from the middle of x, or specifically surrounding $\lfloor \frac{x}{2} \rfloor$.



Hash Cryptography in Cryptocurrencies

So now that we know the fundamentals of hash functions, what kind of hash functions do you think are used in cryptocurrencies?

How complex does the algorithm for the hashing process need to be to be really secure?

SHA-256

You typically see two types of hash cryptography methods used in cryptocurrency, SHA-256 and SHA-512. They perform almost the same calculations when forming a hash code, the primary difference is that SHA-256 has 256 bits in a hash code and SHA-512 has 512 bits in a hash code. The other differences beyond that are the constants the functions use as well.

SHA-256 and SHA-512

SHA-256 works on 256 bits, but traditionally is represented through hexadecimal which will then involve a 64 digit hexadecimal value. By inputting any value, a string can be taken and translated to unicode 8 or ASCII as standard bit forms which is then parsed through the hash function, and then output a 64 digit hashcode in hexadecimal or 256 digits in binary.

Size and Scale of SHA-256

The first element to discuss on SHA-256 is the scope of values to it, which is relevant in the topic of second preimage resistance and collision resistance. SHA-256 accepts inputs up to 2^{64} bits long or 18,446,744,073,709,551,616 bits long.

Converting a Message to Binary

To convert a message in the examples, we will use ASCII 8 bit codes which can be found in the tables at

https://www.rapidtables.com/code/text/ascii-table.html

The expanded tables at this site provide direct binary values for the ASCII Character set for the sake of our examples. We will let

$$Bin_l : M \longrightarrow M_l \tag{2}$$

be our binary conversion function for messages of an arbitrary to a binary message of length l bits.

Converting between Binary and Hexadecimal

Any hexadecimal value is a compressed form of a binary value. Each digit in a hexadecimal value can be represented as a 4 bit value in binary. The following table shows all hexadecimal digits and their equivalent binary values. This will be important since SHA-256 takes in hexadecimal values from it's constants to then output a hash digest in hexadecimal.

0	0000	8	1000
1	0001	9	1001
2	0010	*a*	1010
3	0011	*b*	1011
4	0100	*c*	1100
5	0101	*d*	1101
6	0110	*e*	1110
7	0111	*f*	1111

Constants: K_0 to K_{63}

The first set of constants we establish is a collection of 64 constants in an array labeled K. Each K_i is defined by the first 32 digits of the binary representation of the fractional portion of the cube root of the ith prime. More easily stated, to obtain K_i, we take p_i to be the ith prime to occur in the integers. Then we have $\sqrt[3]{p_i} = q_i + r_i$ with $q_i \in \mathbb{Z}$ and $r_i \in \mathbb{R}$ and $0 \leq r_i < 1$. r_i is now the fractional part of the result of $\sqrt[3]{p_i}$. By converting r_i to binary, we then take the first 32 digits to define K_i, or $Bin(r_i) = K_i$.

Constants: K_0 to K_{63}

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Constants: $H_0^{(0)}$ to $H_7^{(0)}$

The next set of constants we define is our initial hash values. There are 8 values to be defined. These values are defined in the same manner as the values of K_i, however $H_i^{(0)}$ is defined by square rooting the ith prime. The values of $H_i^{(0)}$ are as listed below.

$H_0^{(0)}$ =6a09e667 $H_1^{(0)}$ =bb67ae85 $H_2^{(0)}$ =3c6ef372 $H_3^{(0)}$ =a54ff53a
$H_4^{(0)}$ =510e527f $H_5^{(0)}$ =9b05688c $H_6^{(0)}$ =1f83d9ab $H_7^{(0)}$ =5be0cd19

Boolean Algebra

To proceed to the functions of the SHA-256 algorithm, we need to observe 3 functions of boolean algebra. These functions are

AND	$x \wedge y$
OR	$x \vee y$
XOR	$x \oplus y$
NOT	$\neg x$

Boolean Algebra

AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Tabl

A	B
0	1
1	0

Boolean Algebra

Lastly we need the concept of $R = X \oplus Y \oplus Z$. The following array details this case

X	Y	Z	R
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Shifts and Rotations

Shifts and rotations are direct manipulations of binary values used in the defined equations for SHA-256. For both of these functions, let x be an l-bit binary value. First we define shifting as follows

$$SHR^n(x) = x >> n \quad \text{for } 0 \leq n < l \tag{3}$$

$$SHL^n(x) = x << n \quad \text{for } 0 \leq n < l \tag{4}$$

While this is the formal definition, it doesn't explain much unless your familiar with the operation. For example, let x be an 8-bit binary value given by 11010101, and let $n = 5$. Then $SHR_n(x) = 00000110$ since we shifted x to the right by 5. If we did a left shift instead, we would have 10100000.

Shifts and Rotations

Next we define rotations. Unlike shifts which turn portions of a binary value into 0's, a rotation wraps the binary values back onto the other end. We define a rotation to the right as

$$ROTR^n(x) = (SHR^n(x)) \vee (SHL^{l-n}(x)) \quad \text{for } 0 \leq n < l \tag{5}$$

Using our previous example again, for a rotation by 5, we would have $ROTR_5(x) = 00000110 \vee 10101000 = 10101110$.

Choose Function

Next we define functions used in the algorithm. The first is the choose function as follows

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{6}$$

This functions uses x to choose whether to output the value of y or z, as given by

$$\mathrm{Ch}(x, y, z) = \begin{cases} y & \text{if} \quad x = 1 \\ z & \text{if} \quad x = 0 \end{cases} \tag{7}$$

Example: x=1001, y=1010, z=0010. Then

$$Ch(x, y, z) = (1001 \wedge 1010) \oplus (0110 \wedge 0010) = (1000) \oplus (0010) = 1010$$

Majority Function

The next function is the majority function

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{8}$$

Where whichever value has the majority is the result

Example: x=1001, y=1010, z=0010. Then

$$Maj(x, y, z) = (1001 \wedge 1010) \oplus (1001 \wedge 0010) \oplus (1010 \wedge 0010) =$$

$$(1000) \oplus (0000) \oplus (0010) = 1010$$

$$\begin{aligned} x &= 1001 \\ y &= 1010 \\ z &= 0010 \\ Maj(x, y, z) &= 1010 \end{aligned}$$

Functions

$$\sigma_0 = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \tag{9}$$

$$\sigma_1 = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \tag{10}$$

Example of σ_0, let $x = 11011001101011010100101100101000$

$$\begin{aligned} ROTR^7(x) &= 01010001101100110101101010010110 \\ ROTR^{18}(x) &= 01010010110010100011011001101011 \\ SHR^3(x) &= 00011011001101011010100101100101 \\ \sigma_0(x) &= 00011000010011001100010110011000 \end{aligned}$$

Functions

$$\Sigma_0 = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{11}$$

$$\Sigma_1 = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{12}$$

Example of Σ_0, let $x = 11011001101011010100101100101000$

$$\begin{aligned} ROTR^2(x) &= 00110110011010110101001011001010 \\ ROTR^{13}(x) &= 01011001010001101100110101101010 \\ ROTR^{22}(x) &= 10110101001011001010001101100110 \\ \Sigma_0(x) &= 11011010000000100111100110000110 \end{aligned}$$

SHA-256 Preprocessing

In preprocessing for this hash function, we need to convert our message to binary and then pad it and parse. We start with the padding of our message.

Let M be our message and $Bin_l(M)$ be our binary conversion of M where l is the least bits we can use to represent M. To pad our message we have $Pad(M) = Bin_l(M)||1||Bin_k(0)||Bin_{64}(l)$ where we use $||$ to represent appending terms of bits together(ex: $0110||1010 = 01101010$). k is given to us by $l + 1 + k \equiv 448 (\text{mod } 512)$. So we're taking our message M in binary and appending a 1, then appending k zeroes, and finally appending the length of our binary message as a 64 bit value. So our padded value is now a multiple of 512 as a result of the modulus dictating the value of k.

SHA-256 Preprocessing

The next step in preprocessing will be parsing our padded message. As shown previously, the length of our padded message is a multiple of 512, suppose $512n$. In parsing the message, we split $Pad(M)$ into 512 bit blocks, labelled $M^{(1)}, M^{(2)}, ..., M^{(n)}$. In each of the blocks, we further divided the 512 bits into 16 32-bit words labelled as $M_0^{(i)}, M_1^{(i)}, ..., M_{15}^{(i)}$ for the ith block of $Pad(M)$. So when fully parsed, $Pars(Pad(M))$ is turned into $M_0^{(1)}, M_1^{(1)}, ..., M_{15}^{(1)}, M_0^{(2)}, ..., M_{15}^{(2)}, ..., M_0^{(n)}, ..., M_{15}^{(n)}$.

SHA-256 Algorithm

Now with M converted, padded, and parsed, the set of words given by $M_0^{(1)}, M_1^{(1)}, ..., M_{15}^{(1)}, M_0^{(2)}, ..., M_{15}^{(2)}, ..., M_0^{(n)}, ..., M_{15}^{(n)}$ can be processed. With this we can now define the variables in our algorithm and begin processing M into a hash digest. We start by defining W_t(t-steps per block) as follows

$$W_t = \begin{cases} M_t^{(i)} & \text{for } 0 \leq t < 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & \text{for } 16 \leq t \leq 63 \end{cases} \tag{13}$$

SHA-256 Algorithm

Next is implementing our initial hash values for the block we're processing. Note that our initial hash values defined in our constants before are used in the case of $i = 1$. These will be defined for the ith block as

$$a = H_0^{(i-1)}$$
$$b = H_1^{(i-1)}$$
$$c = H_2^{(i-1)}$$
$$d = H_3^{(i-1)}$$
$$e = H_4^{(i-1)}$$
$$f = H_5^{(i-1)}$$
$$g = H_6^{(i-1)}$$
$$h = H_7^{(i-1)}$$

SHA-256 Algorithm

Now with W_t defined we can proceed to the portion of processing a block. For $t = 0$ to $t = 63$ we process the following equations(note that all addition is modulo 2^{32})

$$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t$$
$$T_2 = \Sigma_0(a) + Maj(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

SHA-256 Algorithm

Now that we've shuffled $a, \ldots, h$ significantly by performing the step 64 times, we use this last set of equations to update our hash values

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)} \\
H_5^{(i)} &= f + H_5^{(i-1)} \\
H_6^{(i)} &= g + H_6^{(i-1)} \\
H_7^{(i)} &= h + H_7^{(i-1)}
\end{aligned}
$$

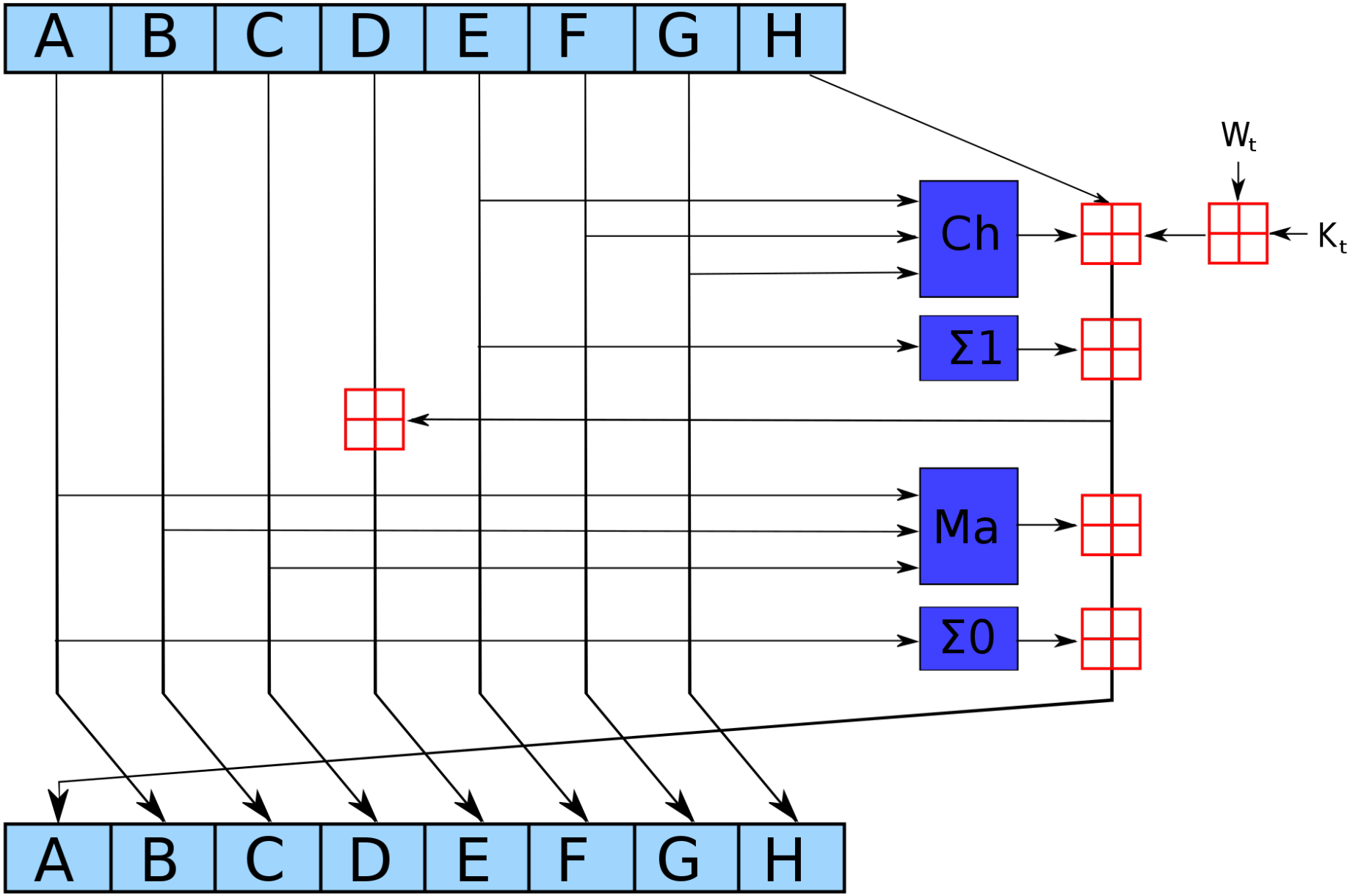
SHA-256 Algorithm

Having completed this set of steps n times for the n blocks we needed to process, we're left with

$$H_0^{(n)}||H_1^{(n)}||H_2^{(n)}||H_3^{(n)}||H_4^{(n)}||H_5^{(n)}||H_6^{(n)}||H_7^{(n)}$$

as our appended hash digest.

SHA-256 Diagram



SHA-256 in Cryptocurrency

When we track back to all the discussion of cryptocurrency, we note that we use a hashed signature on every transaction in a block and on every block to define the next block. This is effectively about 242 hash codes in a single block on the chain. Hashing the approximately 241 transactions is rather simple since we parse the elements of the transaction through the hash function. The tricky part becomes the miners looking for the footer code for the block. Instead of simply parsing a determined term, we are instead searching for one by trying term after term until we find a desired code. This takes more time since we're trying to control 30 of 256 bits. Since we need the first 30 bits as zeroes, with there being 256 bits, there are 2^{256} possible values, and for needing 30 zeros, there are 2^{226} acceptable values, so it is a $\frac{1}{2^{30}}$ chance to find one of the values you're after or $\frac{1}{1073741824}$

Works Cited

"ASCII Table." ASCII Table - ASCII Codes,Hex,Decimal,Binary,Html, https://www.rapidtables.com/code/text/ascii-table.html.

"CS3 Data Structures amp; Algorithms." 10.3. Sample Hash Functions - CS3 Data Structures amp; Algorithms, OpenDSA Project, 21 Aug. 2021, https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/HashFuncExamp.html.

Menezes, A. J., et al. Handbook of Applied Cryptography. CRC, 1996.

Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. 31 Oct. 2008, https://bitcoin.org/bitcoin.pdf.

Works Cited

Romine, Charles H. Secure Hash Standard (SHS) - NIST. Edited by Penny Pritzker and Willie E May, United States of America Department of Commerce, Aug. 2015, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Image Citations

Bitcoin Transaction Diagram
Bitcoin Blockchain Diagram

Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. 31 Oct. 2008, https://bitcoin.org/bitcoin.pdf.

Midsquare Method Example

“CS3 Data Structures amp; Algorithms.” 10.3. Sample Hash Functions - CS3 Data Structures amp; Algorithms, OpenDSA Project, 21 Aug. 2021, https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/HashFuncExamp.html.

Image Citations

Table of K Constants

Romine, Charles H. Secure Hash Standard (SHS) - NIST. Edited by Penny Pritzker and Willie E May, United States of America Department of Commerce, Aug. 2015, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Boolean Algebra Tables

WANG, Tinghui. "Representations of Logic Operations." RealDigital, Real Digital, https://www.realdigital.org/doc/e127ebfa82dbc904b5c0dac5d1adce8e.

Image Citations

SHA-256 Diagram

Kockmeyer. “A Schematic That Shows the SHA-2 Algorithm.” Wikimedia Commons, 22 Mar. 2007, https://commons.wikimedia.org/wiki/File:SHA-2.svg. Accessed 30 Nov. 2021.