Merkle-Patricia Tree

Ledger

Bookkeeping --> accounting --> balance --> state

Bookkeeping is the recording of financial transactions, and is part of the process of accounting in business.[1] Transactions include purchases, sales, receipts and payments by an individual person or an organization/corporation. There are several standard methods of bookkeeping, including the single-entry and double-entry bookkeeping systems.

From <https://en.wikipedia.org/wiki/Bookkeeping>

https://www.dreamstime.com/stock-image-d-life-cycle-accounting-process-illustration-circular-flow-chart-image30625511

Ethereum

IBM Hyperledger Fabric - IBM HF

Authorized capital
Credit
Fixed Assets
Costs
Incomes
Expenses

Op.No.	Input	Output	RemainingAmount
1	123	0	123
2	5	11	117

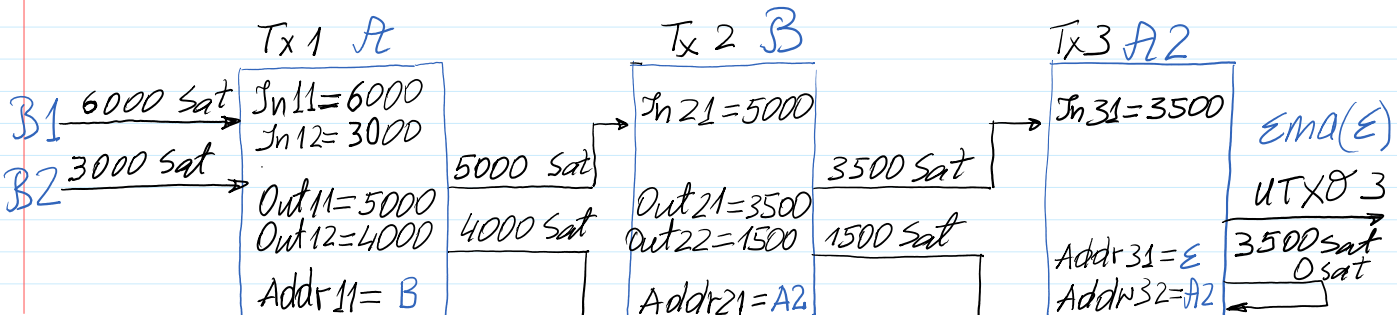
Compare with UTxO system
https://medium.com/@olxc/ethereum-and-smart-contracts-basics-e5c84838b19

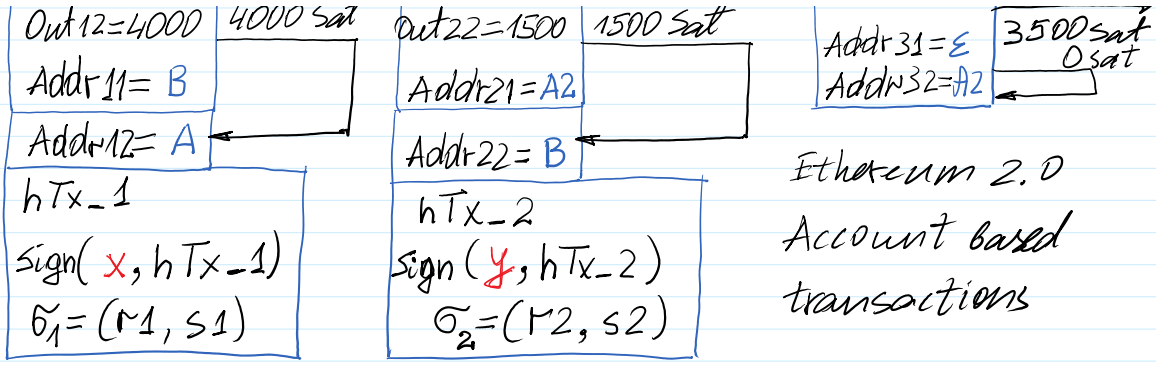
State 1

Authorized Capital	Credit	Fixed Assets	Electricity Cost	Mining	Percent for Credit	Balance

$1\ BTC = 10^{8}\ sat$; $1\ sat = 10^{-8}\ BTC$

Unspent Transactions Output – UTxO





$\Sigma_{In} = \Sigma_{out}$: $Tx1$: $In11 + In12 = Out11 + Out12$

$6000 + 3000 = 5000 + 4000 = 9000$

'$Tx1 : In11 = 6000 || In12 = 3000 || Out11 = 5000 || Out12 = 4000 || Rec1 = B || Rec2 = A$'

$hTx\_1 = h28(\downarrow)$

Transaction template:
Tx_N = 'TxN:In11=…||In12=…||Out11=…||Out12=…||Rec1=…||Rec2=…'

Transactions:
Tx_1 = 'Tx1:In11=6000||In12=3000||Out11=5000||Out12=4000||Rec1=B||Rec2=A'
Tx_2='Tx2:In21=5000||Out21=3500||Out22=1500||Rec1=A2||Rec2=B'
Tx_3='Tx3:In31=3500||Out31=3500||Out32=0||Rec1=E||Rec2=A2'

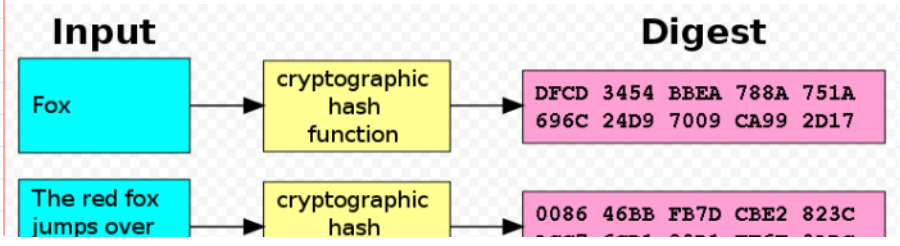
>> hTx_1=h28(Tx_1)
hTx_1 = AFC73D8

>> hTx_2=h28('Tx2:In21=5000||Out21=3500||Out22=1500||Rec1=A2||Rec2=B')
>> hTx_2=h28(Tx_2)
hTx_2 = 13251F8

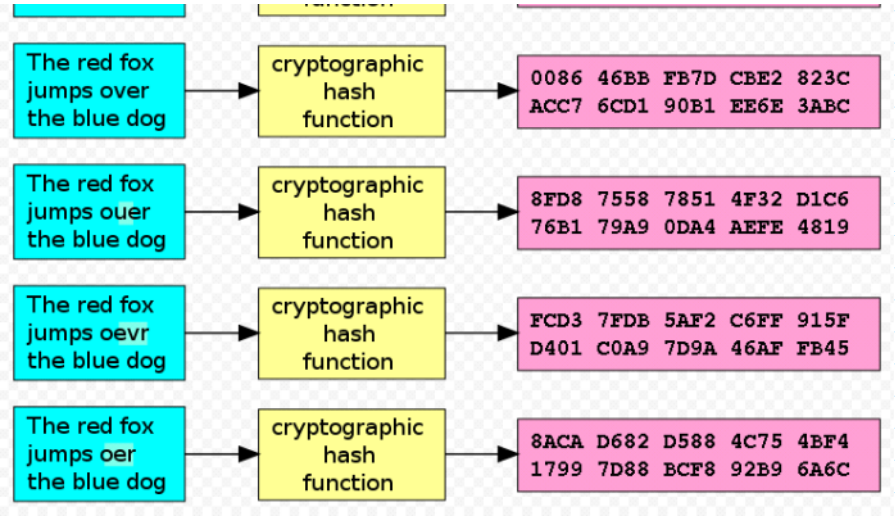
>> hTx_3=h28('Tx3:In31=3500||Out31=3500||Out32=0||Rec1=E||Rec2=A2')
>> hTx_3=h28(Tx_3)
hTx_3 = 99068DE

State tranasition diagramm

H-Functions. Merkle authentication tree



SHA-1 : 160 bits
SHA-256 bits
64 hex



SHA-256 bits
64 hex
Octave 6.3.0

h28('...') - 7 hex
hd28('...') - dec

h24
hd24 hd26 hd28
sha256 AES128

Merkle_Tree

Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone

Binary trees

A *binary tree* is a structure consisting of vertices and directed edges. The vertices are divided into three types:

1. a *root vertex*. The root has two edges directed towards it, a left and a right edge.
2. *internal vertices*. Each internal vertex has three edges incident to it – an upper edge directed away from it, and left and right edges directed towards it.
3. *leaves*. Each leaf vertex has one edge incident to it, and directed away from it.

The vertices incident with the left and right edges of an internal vertex (or the root) are called the *children* of the internal vertex. The internal (or root) vertex is called the *parent* of the associated children. Figure 13.5 illustrates a binary tree with 7 vertices and 6 edges.

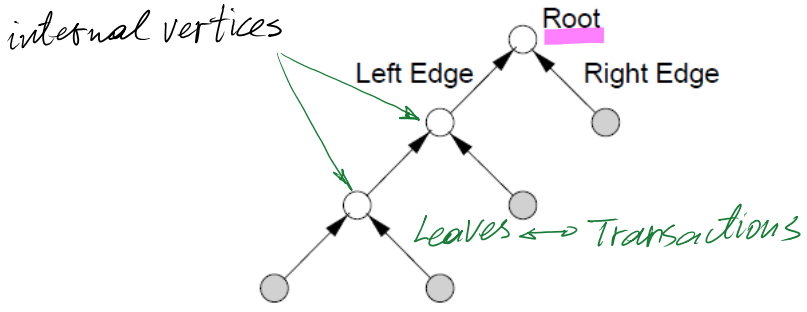


Figure 13.5: *A binary tree (with 4 shaded leaves and 3 internal vertices).*

Constructing and using authentication trees

Consider a binary tree T which has t leaves. Let h be a collision-resistant hash function. T can be used to authenticate t public values, $Y_1, Y_2, \ldots, Y_t$, by constructing an *authentication tree* T^* as follows.

1. Label each of the t leaves by a unique public value Y_i.
2. On the edge directed away from the leaf labeled Y_i, put the label $h(Y_i)$.
3. If the left and right edge of an internal vertex are labeled h_1 and h_2, respectively, label the upper edge of the vertex $h(h_1 \| h_2)$.
4. If the edges directed toward the root vertex are labeled u_1 and u_2, label the root vertex $h(u_1 \| u_2)$.

Once the public values are assigned to leaves of the binary tree, such a labeling is well-defined. Figure 13.6 illustrates an authentication tree with 4 leaves. Assuming some means to authenticate the label on the root vertex, an authentication tree provides a means to authenticate any of the t public leaf values Y_i, as follows. For each public value Y_i, there is a unique path (the *authentication path*) from Y_i to the root. Each edge on the path is a left or right edge of an internal vertex or the root. If e is such an edge directed towards vertex x, record the label on the other edge (not e) directed toward x. This sequence of labels (the *authentication path values*) used in the correct order provides the authentication of Y_i, as illustrated by Example 13.17. Note that if a single leaf value (e.g., Y_1) is altered, maliciously or otherwise, then authentication of that value will fail.

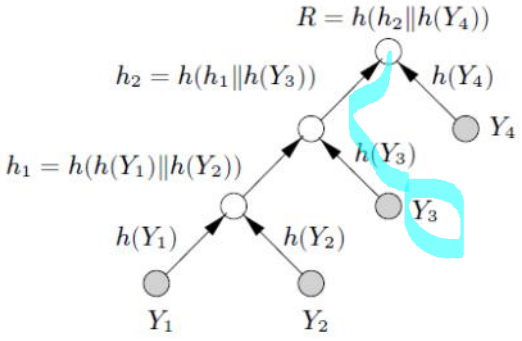


Figure 13.6: *An authentication tree.*

13.17 Example (*key verification using authentication trees*) Refer to Figure 13.6. The public value Y_1 can be authenticated by providing the sequence of labels $h(Y_2)$, $h(Y_3)$, $h(Y_4)$. The authentication proceeds as follows: compute $h(Y_1)$; next compute $h_1 = h(h(Y_1)) \| h(Y_2))$; then compute $h_2 = h(h_1 \| h(Y_3))$; finally, accept Y_1 as authentic if $h(h_2 \| h(Y_4)) = R$, where the root value R is known to be authentic. □

The advantage of authentication trees is evident by considering the storage required to allow authentication of t public values using the following (very simple) alternate approach: an entity A authenticates t public values $Y_1, Y_2, \ldots, Y_t$ by registering each with a trusted third party. This approach requires registration of t public values, which may raise storage issues at the third party when t is large. In contrast, an authentication tree requires only a single value be registered with the third party.

If a public key Y_i of an entity A is the value corresponding to a leaf in an authentication tree, and A wishes to provide B with information allowing B to verify the authenticity of Y_i, then A must (store and) provide to B both Y_i and all hash values associated with the authentication path from Y_i to the root; in addition, B must have prior knowledge and trust in the authenticity of the root value R. These values collectively guarantee authenticity, analogous to the signature on a public-key certificate. The number of values each party must store (and provide to others to allow verification of its public key) is $\lg(t)$, as per Fact 13.19.

13.18 Fact (*depth of a binary tree*) Consider the length of (or number of edges in) the path from each leaf to the root in a binary tree. The length of the longest such path is minimized when the tree is *balanced*, i.e., when the tree is constructed such that all such paths differ in length by at most one. The length of the path from a leaf to the root in a balanced binary tree containing t leaves is about $\lg(t)$.

13.19 Fact (*length of authentication paths*) Using a balanced binary tree (Fact 13.18) as an authentication tree with t public values as leaves, authenticating a public value therein may be achieved by hashing $\lg(t)$ values along the path to the root.

13.20 Remark (*time-space tradeoff*) Authentication trees require only a single value (the root value) in a tree be registered as authentic, but verification of the authenticity of any particular leaf value requires access to and hashing of all values along the authentication path from leaf to root.

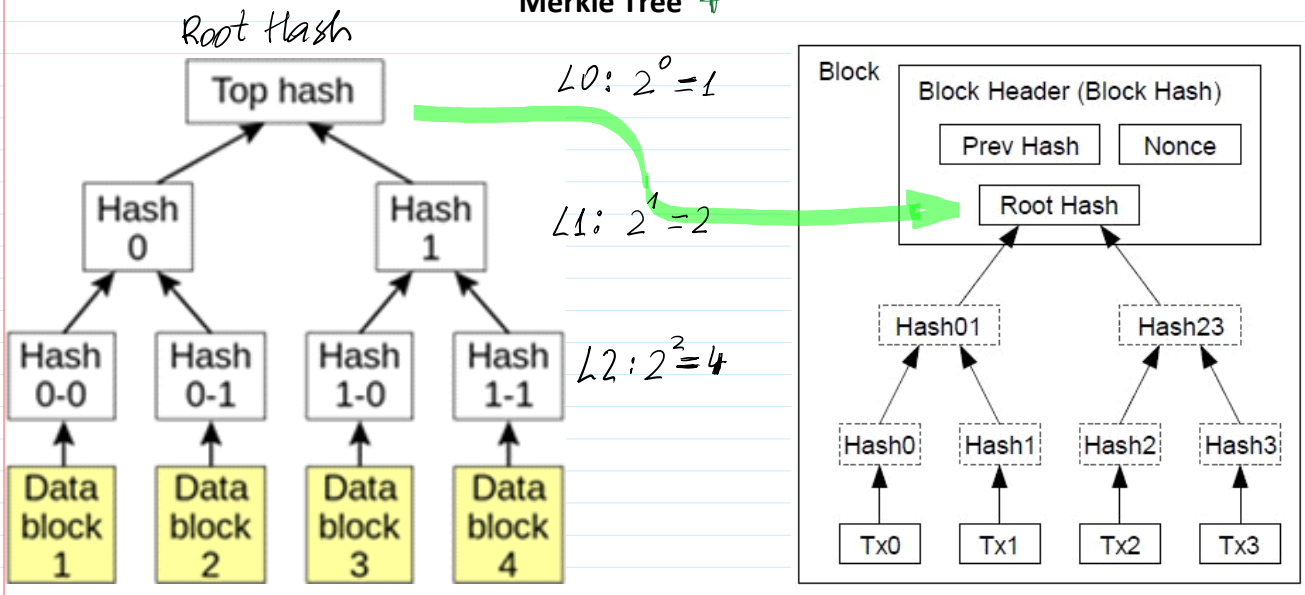
13.21 Remark (*changing leaf values*) To change a public (leaf) value or add more values to an authentication tree requires recomputation of the label on the root vertex. For large balanced trees, this may involve a substantial computation. In all cases, re-establishing trust of all users in this new root value (i.e., its authenticity) is necessary.

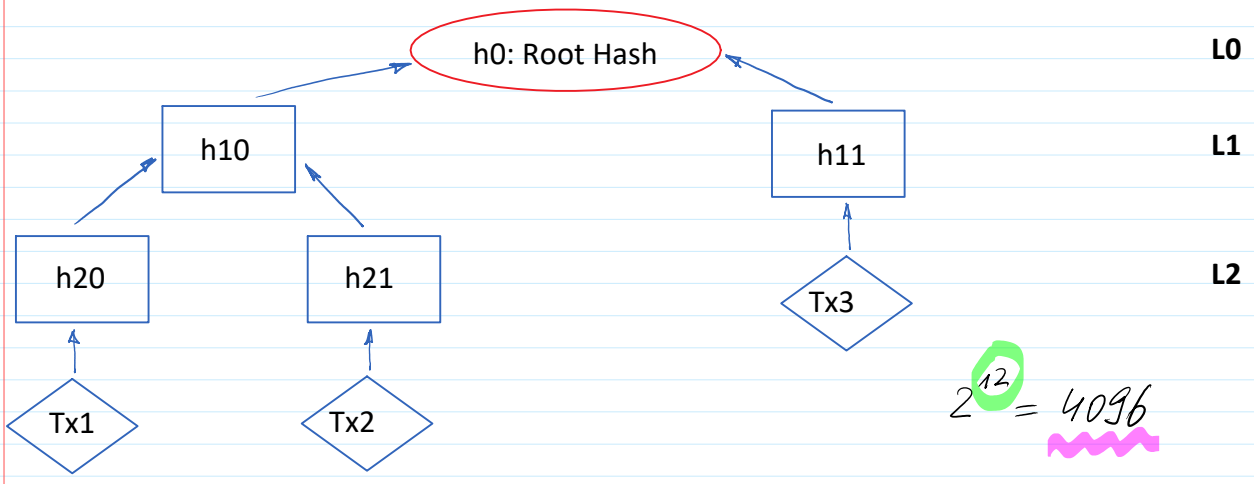
The computational cost involved in adding more values to a tree (Remark 13.21) may motivate constructing the new tree as an unbalanced tree with the new leaf value (or a subtree of such values) being the right child of the root, and the old tree, the left. Another motivation for allowing unbalanced trees arises when some leaf values are referenced far more frequently than others.

Bitcoin transactions are permanently recorded in the network through files called blocks. Maximum size of the block is currently limited to 1 MB but it may be increased in the future. Each block contains a UNIX time timestamp, which is used in block validity checks to make it more difficult for adversary to manipulate the block chain. New blocks are added to the end of the record (block chain) by referencing the hash of the previous block and once added are never changed. A variable number of transactions is included into a block through the merkle tree (fig 3.). Transactions in the Merkle tree are hashed using double SHA256 (hash of the hash of the transaction message).

Transactions are included into the block's hash indirectly through the merkle root (top hash of a merkle tree). This allows removing old transactions (fig. 4) without modifying the hash of the block. Once the latest transaction is buried under enough blocks, previous transactions serve only as a history of the ownership and can be discarded to save space.

Merkle Tree





>> h20=h28(hTx_1)
h20 = AFC73D8
>> h21=h28(hTx_2)
h21 = 13251F8
>> h11=h28(hTx_3)
h11 = 99068DE

>> h10=h28('AFC73D8||13251F8')
h10 = B7D1B0C

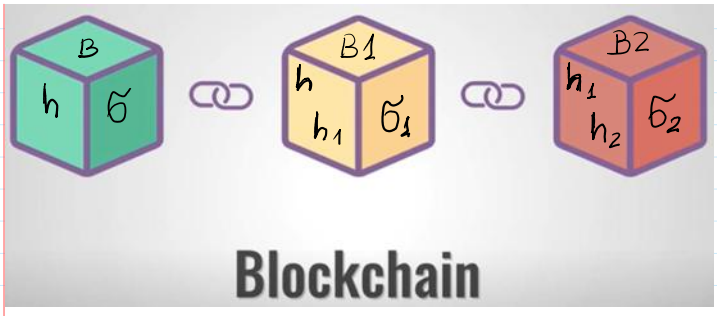
>> h0=h28('B7D1B0C||99068DE')
h0 = **6A34C73**

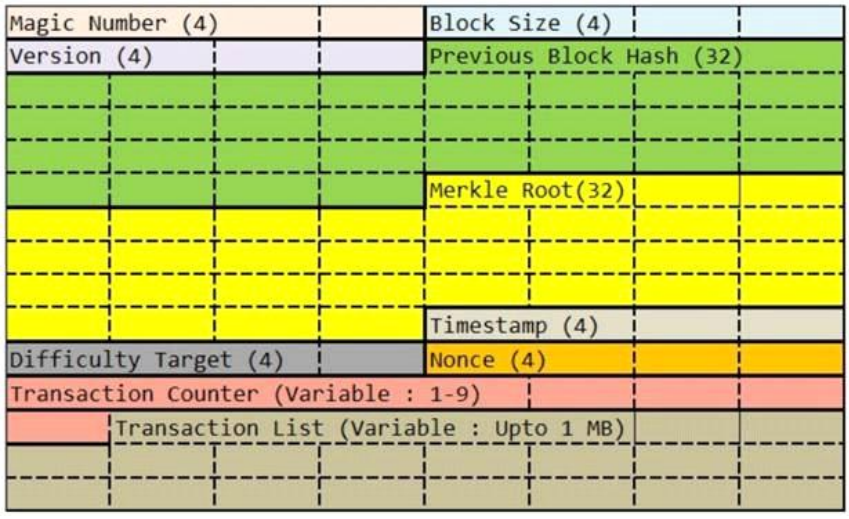
Root Hash: h0

Python : sha256

h20: AFC73D8
h21: 13251F8
h11: 99068DE

h10: B7D1B0C
h0: **6A34C73**





Block size = 4 Bytes
4 Bytes × 8 bits = 32 bits
Block have
$2^{32} - 1 =$ 4294967295
In ASCII encoding
8 bits represents
1 symbol a, b, c, ... #
Block represents
536 870 912 symbols

Difficulty Target (DT): defines the complexity of block mining. In our simulation DT we will choose to find h-value of mining (mined block) having only 1 leading hexadecimal digit equal to 0_h.

h28('RootHash PrevHash 737327631') =
C51E6DE

nonce
>> sha256('RootHash PrevHash 737327631')
ans = F4AE534CD226FAF799 8C8424B348E020BA80639A687E93A0B8C5130ED C51E6DE

>> sha256('RootHash PrevHash 737327632')
ans = B856211DF2EE15E30AB770C1A43CE014ECFE573182AFD885B28D96854DBC5F21
>> sha256('RootHash PrevHash 737327633')
ans = 9C18C764E347A58E57AC3F7A3C2874D5889A0E802699FEA47EEFF8C03BFEDA69
>> sha256('RootHash PrevHash 737327634')
ans = 32B2108A70C39565485CCED9C948E5B7A0027D1EE98642E09D5E4D3D84E16814
>> sha256('RootHash PrevHash 737327635')
ans = A281AC77F5C9AEDEEFFDEDEA85DCEA1C5D76E4222AB80D8A456AEB2AA9EB0F44

$\frac{1}{2^{72}}$

$1\,K = 2^{10}$
$1\,M = 2^{20}$
$1\,G = 2^{30}$
$1\,T = 2^{40}$

>> sha256('RootHash PrevHash 737327648')
ans = 01F9832B2431AFF9D2219E446D613B8361B9903B4B02B8A63990C6B2209785A6

After 17 trials with sequentially increasing nonce Malaga mined a block with DT=1 H-most-significant digit 48-31=17.

>> h28('RootHash PrevHash 737327631')
ans = C51E6DE
>> h28('RootHash PrevHash 737327648')
ans = 09785A6

17 trials again.

DT: to mine a block it is needed to find h-value having leading zero in hexadecimal format: C51E6DE 0XXXXXX

F 1111

$6 \times 4 = 24$ bits

h-value is computed >> h28( ) → 7 hex numbers

What probability to mine a block? Number of 4 bits has $2^4 = 16$ values

0000	0001	0010	0011	...	1001	1010	1011	1100	1101	1110	1111
0	1	2	3		9	10 A	11 B	12 C	13 D	14 E	15 F

The number of possible h-values of 28 bits: 2^{28} >> 2^28 ans = 268 435 456

The number of adequate h-values: 2^{24} >> int64(2^24) ans = 16777216

$$Pr\{\text{to Mine}\} = \frac{2^{24}}{2^{28}} = \frac{1}{2^4} = \frac{1}{16}$$

DT: two leading hex number = 00 00XXXXX

The number of adequate h-values: 2^{20}

$5 \times 4 = 20$

$$Pr\{\text{to Mine}\} = \frac{2^{20}}{2^{28}} = \frac{1}{2^8} = \frac{1}{256}$$

DT: three leading hex number = 000 000XXXX

$4 \times 4 = 16$

$$Pr\{\text{to Mine}\} = \frac{2^{16}}{2^{28}} = \frac{1}{2^{12}} = \frac{1}{4096}$$ >> 2^12 ans = 4096

$$Pr\{\text{to Mine}\} = \frac{1}{2^{28}} = \frac{1}{268\,435\,456}$$ >> 2^28 ans = 268 435 456

The probability to mine a block, e.g. in Bitcoin when DT: is to find SHA256 value having 18 leading zeroes

$1\ Eth = 10^{18}\ Wei$

The probability to mine a block, e.g. in Bitcoin when 1 Eth = 10 Wei

DT : is to find SHA256 value having 18 leading zeroes

Till this place

```
>> sha256('RootHash PrevHash 737327631')
ans = F4AE534CD226FAF799 8C8424B348E020BA80639A687E93A0B8C5130EDC51E6DE
      000000000000000000 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

The number of possible h-values having 256 bits is 2^{256}.

The numbe of adequate h-values of SHA256 is

$256 - 18 \cdot 4 = 256 - 72 = 184$ bits, that are represented 46 hex. num.

The number of adequate values is 2^{184}.

$$Prob\{to\ mine\} = \frac{2^{184}}{2^{256}} = 2^{184-256} = 2^{-72}$$

$2^{72} \sim 4\,GT = 4 \cdot 2^{30} \cdot 2^{40} = 2^2 \cdot 2^{30} \cdot 2^{40} = 2^{72}$.

$1\,K = 2^{10} = 1024$

$1\,M = 2^{20} = \dots$

$1\,G = 2^{30} = \dots$

$1\,T = 2^{40} = \dots$

N = 4 722 366 482 869 645 213 696

Number of trials $N = 1T \cdot 1G \cdot 2^2 = 4 \cdot 2^{40} \cdot 2^{30}$.

Total net capacity $Cap \sim 2000$ Th/sek

$$Time\ T = \frac{N}{Cap} = \frac{4 \cdot 2^{40} \cdot 2^{30}}{2000 \cdot 2^{40}} \approx \frac{4 \cdot 2^{30}}{2^{11}} = 4 \cdot 2^{19}\ s$$

```
>> T=int64(4*2^19)
T = 2097152
>> Tval=T/3600
Tval = 583
>> Tdien=Tval/24
Tdien = 24
```

Private blockchain ⟷ Public blockchain

Monero blockchain : Transactions sums → confidential → verifiable

Sender, Receiver → anonymous

How to realize confidential & verifiable transactions.

hPrBl	hRoot	Bl_N:hPrBl=0CAF06F||hRoot=2CC219F||hTx_N1=AFC73D8||hTx_N2=13251F8||hTx_N3=5B5412B||Nonce=1000	hBl_N	Nonce	hBl_N_Mined
0CAF06F	2CC219F	**Bl_1:hPrBl=0CAF06F||hRoot=**2CC219F||**hTx_1=**AFC73D8||**hTx_2=**13251F8||**hTx_3=**5B5412B||**Nonce=1000**		1021	**06F61B0**