ABSTRACT:

Year after Year, the exponentially increasing crime occurs in the tech world either through direct funds theft like bitcoin/NFT/digital Art theft or through Spam DDoS-Attacks on sensitive sites from Bots.

This report shows a method to convert a picture into a hash code using a hashing sequence similar to SHA256 where the image is converted to black and white then to binary then to a small unique hash value according to the algorithm designed by us.
This is the foundational algorithm and mechanism to how blockchain works

Image hashing is done for two reasons:

1. Digital Art/NFT verification using image hashing
2. New Captcha method to prevent Malicious Auto-Bots using picture check

Index

EXAMPLE

Upload the image in the Anvil Web App

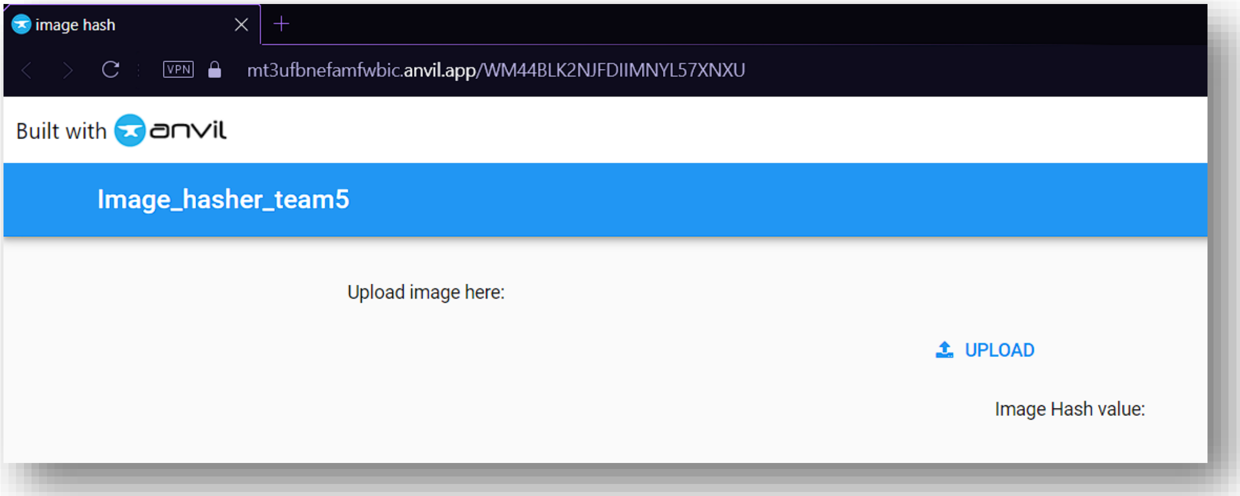


Image Hash value:

Output:

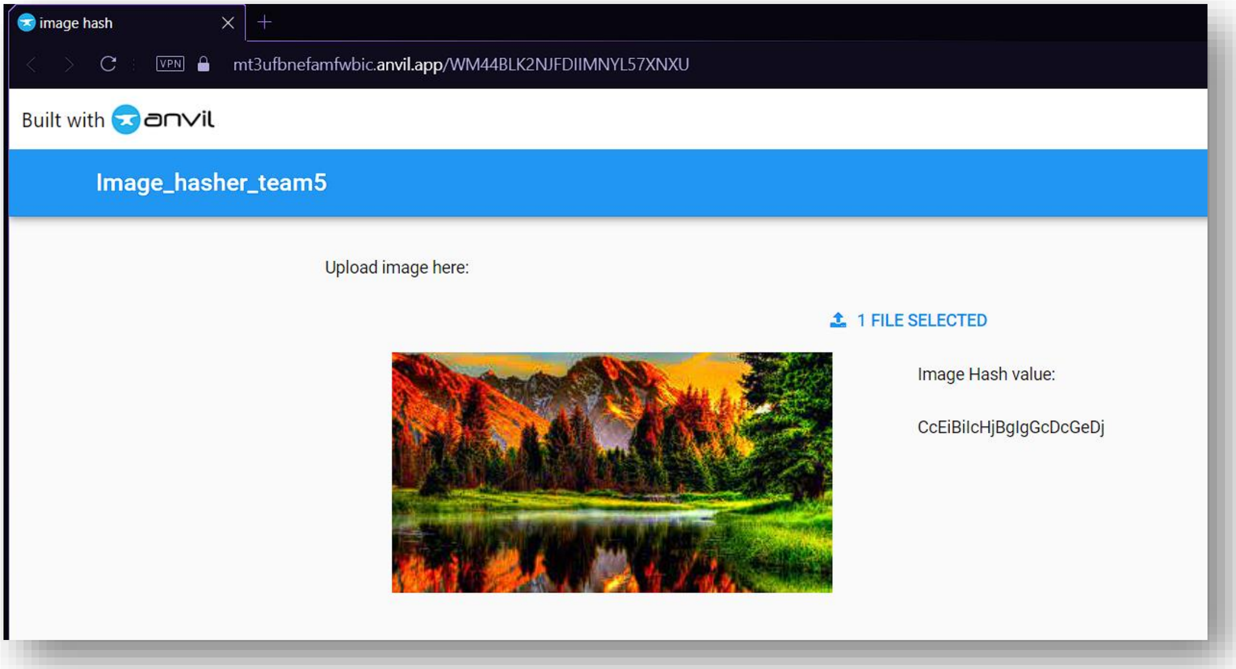
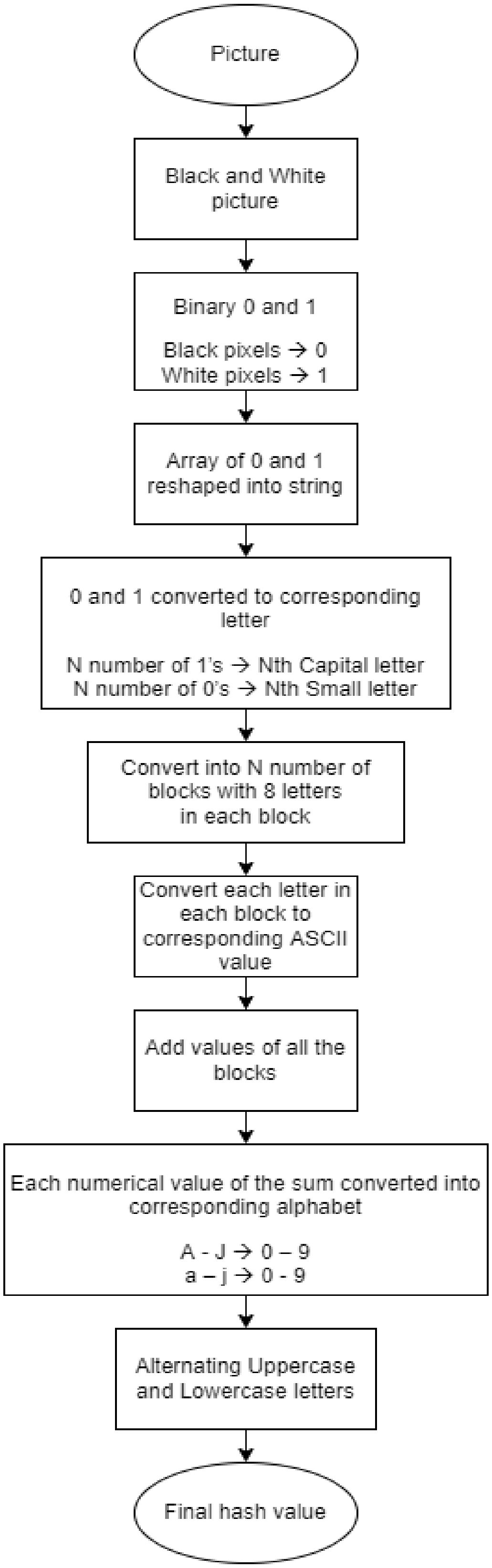


Image Hash value:

CcEiBiIcHjBgIgGcDcGeDj

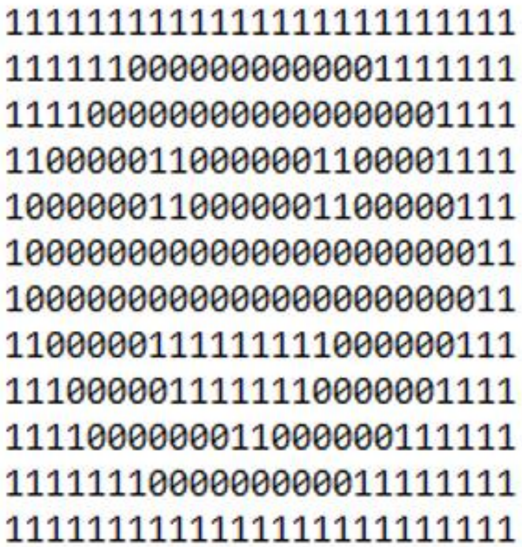
FLOWCHART


Picture
Black and White picture
Binary 0 and 1
Black pixels → 0
White pixels → 1
Array of 0 and 1 reshaped into string
0 and 1 converted to corresponding letter
N number of 1's → Nth Capital letter
N number of 0's → Nth Small letter
Convert into N number of blocks with 8 letters in each block
Convert each letter in each block to corresponding ASCII value
Add values of all the blocks
Each numerical value of the sum converted into corresponding alphabet
A - J → 0 – 9
a – j → 0 - 9
Alternating Uppercase and Lowercase letters
Final hash value

EXPLANATION

Step 1: Image gets converted to greyscale and then to binary values of 0s & 1s where

0 → black
1 → white



```
11111111111111111111111111
11111100000000000001111111
11110000000000000000001111
11000000110000001100001111
10000000110000001100000111
10000000000000000000000011
10000000000000000000000011
11000000111111111000000111
11100000011111110000001111
11110000000011000000111111
11111110000000000011111111
11111111111111111111111111
```

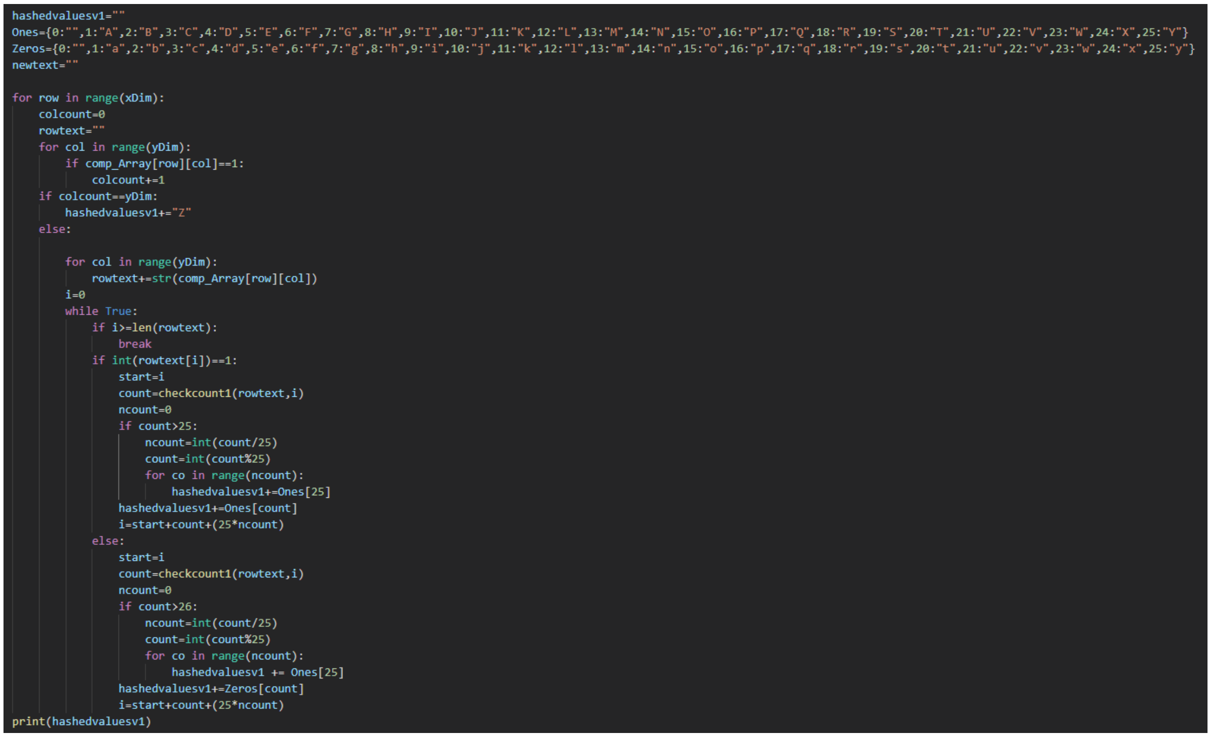
Step 2:
The array of 0s and 1s is reshaped to form a long string

```
1111111111111111111111111
1111110000000000001111111
1111000000000000000001111
1100000110000001100001111
1000000110000001100000111
1000000000000000000000011
1000000000000000000000011
1100000111111110000000111
1110000011111100000001111
1111000000011000000111111
1111111000000000011111111
1111111111111111111111111
```

```
1111111111111111111111111111111000000000000111111111110000000000000000011111100000110000001100001111
1000000110000001100000111100000000000000000000001110000000000000000000000111100000111111110000000111
1110000011111100000001111111100000001100000011111111111100000000001111111111111111111111111111111111
```

Step 3: The 0s & 1s are converted to the corresponding letter using the following logic

111111111111 → 12 x 1s → 12th capital letter → L
0000000 → 7 x 0s → 7th Small letter → g
This is done by:
count the number of 1s and convert it to corresponding uppercase letter
count the number of 0s and convert it to corresponding lowercase letter

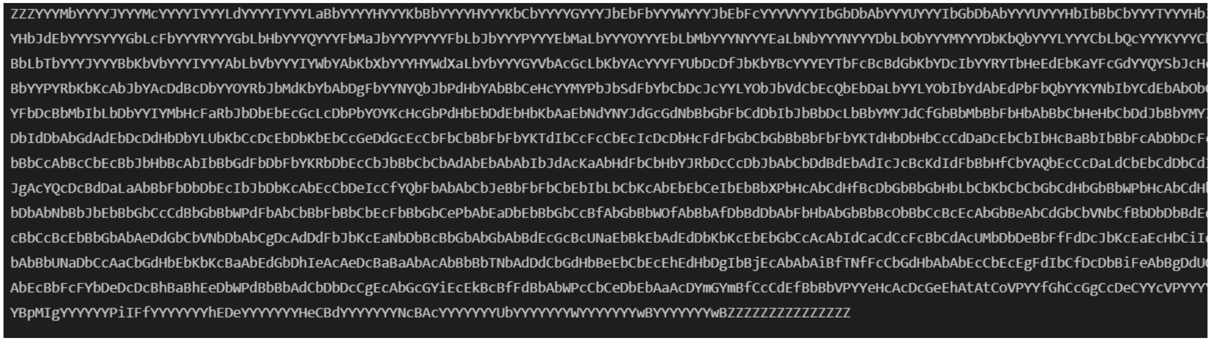


```
hashedvaluesv1=""
Ones={0:"",1:"A",2:"B",3:"C",4:"D",5:"E",6:"F",7:"G",8:"H",9:"I",10:"J",11:"K",12:"L",13:"M",14:"N",15:"O",16:"P",17:"Q",18:"R",19:"S",20:"T",21:"U",22:"V",23:"W",24:"X",25:"Y"}
Zeros={0:"",1:"a",2:"b",3:"c",4:"d",5:"e",6:"f",7:"g",8:"h",9:"i",10:"j",11:"k",12:"l",13:"m",14:"n",15:"o",16:"p",17:"q",18:"r",19:"s",20:"t",21:"u",22:"v",23:"w",24:"x",25:"y"}
newtext=""

for row in range(xDim):
    colcount=0
    rowtext=""
    for col in range(yDim):
        if comp_Array[row][col]==1:
            colcount+=1
    if colcount==yDim:
        hashedvaluesv1+="Z"
    else:

        for col in range(yDim):
            rowtext+=str(comp_Array[row][col])
        i=0
        while True:
            if i>=len(rowtext):
                break
            if int(rowtext[i])==1:
                start=i
                count=checkcount1(rowtext,i)
                ncount=0
                if count>25:
                    ncount=int(count/25)
                    count=int(count%25)
                    for co in range(ncount):
                        hashedvaluesv1+=Ones[25]
                hashedvaluesv1+=Ones[count]
                i=start+count+(25*ncount)
            else:
                start=i
                count=checkcount1(rowtext,i)
                ncount=0
                if count>26:
                    ncount=int(count/25)
                    count=int(count%25)
                    for co in range(ncount):
                        hashedvaluesv1 += Ones[25]
                hashedvaluesv1+=Zeros[count]
                i=start+count+(25*ncount)
print(hashedvaluesv1)
```

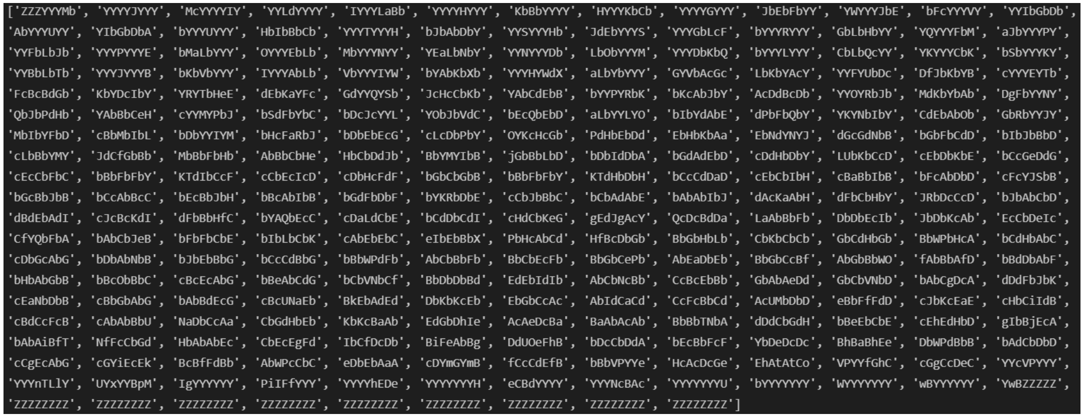
Corresponding output:



```
ZZZYYYMbYYYYJYYYMcYYYYIYYYLdYYYYIYYYLaBbYYYYHYYYKbBbYYYYHYYYKbCbYYYYGYYYJbEbFbYYYWYYYJbEbFcYYYVYYYIbGbDbAbYYYUYYYIbGbDbAbYYYUYYYHbIbBbCbYYYTYYYHb
YHbJdEbYYYSYYYGbLcFbYYYRYYYGbLbHbYYYQYYYFbMaJbYYYPYYYFbLbJbYYYPYYYEbMaLbYYYOYYYEbLbMbYYYNYYYEaLbNbYYYNYYYDbLbObYYYMYYYDbKbQbYYYLYYYCbLbQcYYYKYYYC
BbLbTbYYYJYYYBbKbVbYYYIYYYAbLbVbYYYIYWbYAbKbXbYYYHYWdXaLbYbYYYGYVbAcGcLbKbYAcYYYFYUbDcDfJbKbYBcYYYEYTbFcBcBdGbKbYDcIbYYRYTbHeEdEbKaYFcGdYYQYSbJcH
BbYYPYRbKbKcAbJbYAcDdBcDbYYOYRbJbMdKbYbAbDgFbYYNYQbJbPdHbYAbBbCeHcYYMYPbJbSdFbYbCbDcJcYYLYObJbVdCbEcQbEbDaLbYYLYObIbYdAbEdPbFbQbYYKYNbIbYCdEbAbOb
YFbDcBbMbIbLbDbYYIYMbHcFaRbJbDbEbEcGcLcDbPbYOYKcHcGbPdHbEbDdEbHbKbAaEbNdYNYJdGcGdNbBbGbFbCdDbIbJbBbDcLbBbYMYJdCfGbBbMbBbFbHbAbBbCbHeHbCbDdJbBbYMY
DbIdDbAbGdAdEbDcDdHbDbYLUbKbCcDcEbDbKbEbCcGeDdGcEcCbFbCbBbFbFbYKTdIbCcFcCbEcIcDcDbHcFdFbGbCbGbBbBbFbFbYKTdHbDbHbCcCdDaDcEbCbIbHcBaBbIbBbFcAbDbDcF
bBbCcAbBcCbEcBbJbHbBcAbIbBbGdFbDbFbYKRbDbEcCbJbBbCbCbAdAbEbAbAbIbJdAcKaAbHdFbCbHbYJRbDcCcDbJbAbCbDdBdEbAdIcJcBcKdIdFbBbHfCbYAQbEcCcDaLdCbEbCdDbCd
JgAcYQcDcBdDaLaAbBbFbDbDbEcIbJbDbKcAbEcCbDeIcCfYQbFbAbAbCbJeBbFbFbCbEbIbLbCbKcAbEbEbCeIbEbBbXPbHcAbCdHfBcDbGbBbGbHbLbCbKbCbCbGbCdHbGbBbWPbHcAbCdH
bDbAbNbBbJbEbBbGbCcCdBbGbBbWPdFbAbCbBbFbBbCbEcFbBbGbCePbAbEaDbEbBbGbCcBfAbGbBbWOfAbBbAfDbBdDbAbFbHbAbGbBbBcObBbCcBcEcAbGbBeAbCdGbCbVNbCfBbDbDbBdE
cBbCcBcEbBbGbAbAeDdGbCbVNbDbAbCgDcAdDdFbJbKcEaNbDbBcBbGbAbGbAbBdEcGcBcUNaEbBkEbAdEdDbKbKcEbEbGbCcAcAbIdCaCdCcFcBbCdAcUMbDbDeBbFfFdDcJbKcEaEcHbCiI
bAbBbUNaDbCcAaCbGdHbEbKbKcBaAbEdGbDhIeAcAeDcBaBaAbAcAbBbBbTNbAdDdCbGdHbBeEbCbEcEhEdHbDgIbBjEcAbAbAiBfTNfFcCbGdHbAbAbEcCbEcEgFdIbCfDcDbBiFeAbBgDdU
AbEcBbFcFYbDeDcDcBhBaBhEeDbWPdBbBbAdCbDbDcCgEcAbGcGYiEcEkBcBfFdBbAbWPcCbCeDbEbAaAcDYmGYmBfCcCdEfBbBbVPYYeHcAcDcGeEhAtAtCoVPYYfGhCcGgCcDeCYYcVPYYY
YBpMIgYYYYYYPiIFfYYYYYYYhEDeYYYYYYYHeCBdYYYYYYYNcBAcYYYYYYYUbYYYYYYYWYYYYYYYwBYYYYYYYwBZZZZZZZZZZZZZZZ
```

Step 4: Create a Hashing algorithm to convert this string into a hash value

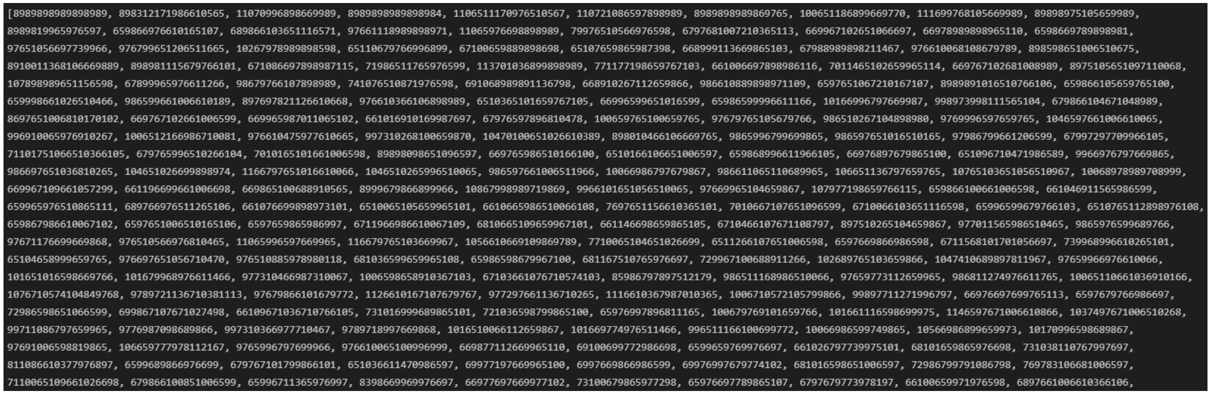
1. split the long string into N blocks of 8 characters



```
['ZZZYYYMb', 'YYYYJYYY', 'McYYYYIY', 'YYLdYYYY', 'IYYYLaBb', 'YYYYHYYY', 'KbBbYYYY', 'HYYYKbCb', 'YYYYGYYY', 'JbEbFbYY', 'YWYYYJbE', 'bFcYYYVY', 'YYIbGbDb',
'AbYYYUYY', 'YIbGbDbA', 'bYYYUYYY', 'HbIbBbCb', 'YYYTYYYH', 'bJbAbDbY', 'YYSYYYHb', 'JdEbYYYS', 'YYYGbLcF', 'bYYYRYYY', 'GbLbHbYY', 'YQYYYFbM', 'aJbYYYPY',
'YYFbLbJb', 'YYYPYYYE', 'bMaLbYYY', 'OYYYEbLb', 'MbYYYNYY', 'YEaLbNbY', 'YYNYYYDb', 'LbObYYYM', 'YYYDbKbQ', 'bYYYLYYY', 'CbLbQcYY', 'YKYYYCbK', 'bSbYYYKY',
'YYBbLbTb', 'YYYJYYYB', 'bKbVbYYY', 'IYYYAbLb', 'VbYYYIYW', 'bYAbKbXb', 'YYYHYWdX', 'aLbYbYYY', 'GYVbAcGc', 'LbKbYAcY', 'YYFYUbDc', 'DfJbKbYB', 'cYYYEYTb',
'FcBcBdGb', 'KbYDcIbY', 'YRYTbHeE', 'dEbKaYFc', 'GdYYQYSb', 'JcHcCbKb', 'YAbCdEbB', 'bYYPYRbK', 'bKcAbJbY', 'AcDdBcDb', 'YYOYRbJb', 'MdKbYbAb', 'DgFbYYNY',
'QbJbPdHb', 'YAbBbCeH', 'cYYMYPbJ', 'bSdFbYbC', 'bDcJcYYL', 'YObJbVdC', 'bEcQbEbD', 'aLbYYLYO', 'bIbYdAbE', 'dPbFbQbY', 'YKYNbIbY', 'CdEbAbOb', 'GbRbYYJY',
'MbIbYFbD', 'cBbMbIbL', 'bDbYYIYM', 'bHcFaRbJ', 'bDbEbEcG', 'cLcDbPbY', 'OYKcHcGb', 'PdHbEbDd', 'EbHbKbAa', 'EbNdYNYJ', 'dGcGdNbB', 'bGbFbCdD', 'bIbJbBbD',
'cLbBbYMY', 'JdCfGbBb', 'MbBbFbHb', 'AbBbCbHe', 'HbCbDdJb', 'BbYMYIbB', 'jGbBbLbD', 'bDbIdDbA', 'bGdAdEbD', 'cDdHbDbY', 'LUbKbCcD', 'cEbDbKbE', 'bCcGeDdG',
'cEcCbFbC', 'bBbFbFbY', 'KTdIbCcF', 'cCbEcIcD', 'cDbHcFdF', 'bGbCbGbB', 'bBbFbFbY', 'KTdHbDbH', 'bCcCdDaD', 'cEbCbIbH', 'cBaBbIbB', 'bFcAbDbD', 'cFcYJSbB',
'bGcBbJbB', 'bCcAbBcC', 'bEcBbJbH', 'bBcAbIbB', 'bGdFbDbF', 'bYKRbDbE', 'cCbJbBbC', 'bCbAdAbE', 'bAbAbIbJ', 'dAcKaAbH', 'dFbCbHbY', 'JRbDcCcD', 'bJbAbCbD',
'dBdEbAdI', 'cJcBcKdI', 'dFbBbHfC', 'bYAQbEcC', 'cDaLdCbE', 'bCdDbCdI', 'cHdCbKeG', 'gEdJgAcY', 'QcDcBdDa', 'LaAbBbFb', 'DbDbEcIb', 'JbDbKcAb', 'EcCbDeIc',
'CfYQbFbA', 'bAbCbJeB', 'bFbFbCbE', 'bIbLbCbK', 'cAbEbEbC', 'eIbEbBbX', 'PbHcAbCd', 'HfBcDbGb', 'BbGbHbLb', 'CbKbCbCb', 'GbCdHbGb', 'BbWPbHcA', 'bCdHbAbC',
'cDbGcAbG', 'bDbAbNbB', 'bJbEbBbG', 'bCcCdBbG', 'bBbWPdFb', 'AbCbBbFb', 'BbCbEcFb', 'BbGbCePb', 'AbEaDbEb', 'BbGbCcBf', 'AbGbBbWO', 'fAbBbAfD', 'bBdDbAbF',
'bHbAbGbB', 'bBcObBbC', 'cBcEcAbG', 'bBeAbCdG', 'bCbVNbCf', 'BbDbDbBd', 'EdEbIdIb', 'AbCbNcBb', 'CcBcEbBb', 'GbAbAeDd', 'GbCbVNbD', 'bAbCgDcA', 'dDdFbJbK',
'cEaNbDbB', 'cBbGbAbG', 'bAbBdEcG', 'cBcUNaEb', 'BkEbAdEd', 'DbKbKcEb', 'EbGbCcAc', 'AbIdCaCd', 'CcFcBbCd', 'AcUMbDbD', 'eBbFfFdD', 'cJbKcEaE', 'cHbCiIdB',
'cBdCcFcB', 'cAbAbBbU', 'NaDbCcAa', 'CbGdHbEb', 'KbKcBaAb', 'EdGbDhIe', 'AcAeDcBa', 'BaAbAcAb', 'BbBbTNbA', 'dDdCbGdH', 'bBeEbCbE', 'cEhEdHbD', 'gIbBjEcA',
'bAbAiBfT', 'NfFcCbGd', 'HbAbAbEc', 'CbEcEgFd', 'IbCfDcDb', 'BiFeAbBg', 'DdUOeFhB', 'bDcCbDdA', 'bEcBbFcF', 'YbDeDcDc', 'BhBaBhEe', 'DbWPdBbB', 'bAdCbDbD',
'cCgEcAbG', 'cGYiEcEk', 'BcBfFdBb', 'AbWPcCbC', 'eDbEbAaA', 'cDYmGYmB', 'fCcCdEfB', 'bBbVPYYe', 'HcAcDcGe', 'EhAtAtCo', 'VPYYfGhC', 'cGgCcDeC', 'YYcVPYYY',
'YYYnTLlY', 'UYxYYBpM', 'IgYYYYYY', 'PiIFfYYY', 'YYYYhEDe', 'YYYYYYYH', 'eCBdYYYY', 'YYYNcBAc', 'YYYYYYYU', 'bYYYYYYY', 'WYYYYYYY', 'wBYYYYYY', 'YwBZZZZZ',
'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ', 'ZZZZZZZZ']
```

2. Send the blocks for further hashing by a HashMap

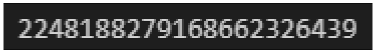
3. Convert each character to its corresponding ascii value
   Corresponding output:



```
[8989898989898989, 898312171986610565, 11070996898669989, 8989898989898984, 1106511170976510567, 110721086597898989, 8989898989869765, 100651186899669770, 111699768105669989, 89898975105659989,
8989819965976597, 659866976610165107, 689866103651116571, 97661118989898971, 11065976698898989, 79976510566976598, 6797681007210365113, 669967102651066697, 66978989898965110, 6598669789898981,
97651056697739966, 976799651206511665, 10267978989898598, 65110679766996899, 67100659889898698, 65107659865987398, 668999113669865103, 67988989898211467, 976610068108679789, 898598651006510675,
8910011368106669889, 898981115679766101, 671086697898987115, 71986511765976599, 113701036899898989, 771177198659767103, 661006697898986116, 7011465102659965114, 669767102681008989, 8975105651097110068,
107898989651156598, 67899965976611266, 98679766107898989, 741076510871976598, 691068989891136798, 668910267112659866, 986610889898971109, 6597651067210167107, 8989891016510766106, 659866105659765100,
659998661026510466, 986599661006610189, 897697821126610668, 976610366106898989, 6510365101659767105, 66996599651016599, 65986599996611166, 10166996797669987, 998973998111565104, 679866104671048989,
8697651006810170102, 669767102661006599, 669965987011065102, 661016910169987697, 67976597896810478, 100659765100659765, 97679765105679766, 986510267104898980, 9769996597659765, 1046597661006610065,
996910065976910267, 1006512166986710081, 976610475977610665, 997310268100659870, 10470100651026610389, 898010466106669765, 9865996799699865, 986597651016510165, 97986799661206599, 67997297709966105,
71101751066510366105, 679765996510266104, 7010165101661006598, 89898098651096597, 669765986510166100, 6510166106651006597, 659868996611966105, 66976897679865100, 651096710471986589, 9966976797669865,
986697651036810265, 104651026699898974, 1166797651016610066, 1046510265996510065, 986597661006511966, 10066986797679867, 986611065110689965, 106651136797659765, 10765103651056510967, 10068978989708999,
669967109661057299, 661196699661006698, 669865100688910565, 8999679866899966, 10867998989719869, 9966101651056510065, 97669965104659867, 107977198659766115, 659866100661006598, 661046911565986599,
659965976510865111, 689766976511265106, 661076699898973101, 6510065105659965101, 6610665986510066108, 7697651156610365101, 7010667107651096599, 6710066103651116598, 65996599679766103, 6510765112898976108,
659867986610067102, 6597651006510165106, 6597659865986997, 6711966986610067109, 6810665109659967101, 661146698659865105, 6710466107671108797, 897510265104659867, 977011565986510465, 9865976599689766,
9767117669669868, 976510566976810465, 11065996597669965, 116679765103669967, 1056610669109869789, 7710065104651026699, 6511266107651006598, 6597669866986598, 6711568101701056697, 739968996610265101,
65104658999659765, 976697651056710470, 976510885978980118, 681036599659965108, 65986598679967100, 681167510765976697, 729967100688911266, 102689765103659866, 1047410689897811967, 97659966976610066,
101651016598669766, 101679968976611466, 977310466987310067, 1006598658910367103, 67103661076710574103, 85986797897512179, 986511168986510066, 97659773112659965, 986811274976611765, 10065110661036910166,
1076710574104849768, 9789721136710381113, 97679866101679772, 1126610167107679767, 977297661136710265, 1116610367987010365, 1006710572105799866, 99897711271996797, 66976697699765113, 6597679766986697,
72986598651066599, 699867107671027498, 66109671036710766105, 731016999689865101, 721036598799865100, 65976997896811165, 100679769101659766, 101661116598699975, 1146597671006610866, 1037497671006510268,
99711086797659965, 9776987098689866, 997310366977710467, 9789718997669868, 1016510066112659867, 101669774976511466, 996511166100699772, 10066986599749865, 10566986899659973, 10170996598689867,
97691006598819865, 106659777978112167, 9765996797699966, 976610065100996999, 669877112669965110, 69100699772986698, 6599659769976697, 661026797739975101, 68101659865976698, 731038110767997697,
811086610377976897, 6599689866976699, 679767101799866101, 651036611470986597, 69977197669965100, 6997669866986599, 69976997679774102, 681016598651006597, 72986799791086798, 769783106681006597,
7110065109661026698, 679866100851006599, 65996711365976997, 8398669969976697, 66977697669977102, 73100679865977298, 65976697789865107, 6797679773978197, 66100659971976598, 6897661006610366106,
```

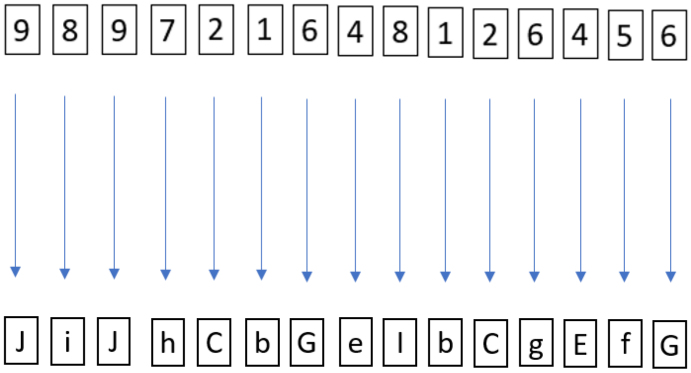
4. add all N blocks of ascii values to get one number

   Corresponding output:

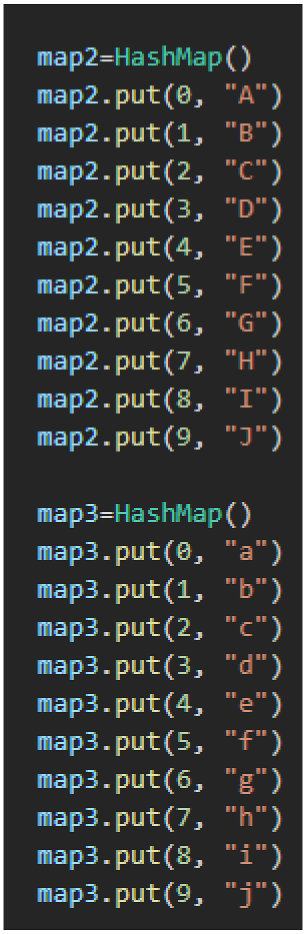


```
2248188279168662326439
```

5. convert the sum → letters according to a Hash map A-I
   example:



Alternate Capital & Small



```
map2=HashMap()
map2.put(0, "A")
map2.put(1, "B")
map2.put(2, "C")
map2.put(3, "D")
map2.put(4, "E")
map2.put(5, "F")
map2.put(6, "G")
map2.put(7, "H")
map2.put(8, "I")
map2.put(9, "J")

map3=HashMap()
map3.put(0, "a")
map3.put(1, "b")
map3.put(2, "c")
map3.put(3, "d")
map3.put(4, "e")
map3.put(5, "f")
map3.put(6, "g")
map3.put(7, "h")
map3.put(8, "i")
map3.put(9, "j")
```

6. even values become capital letters

7. odd values become small letters

Step 5: show final hash value

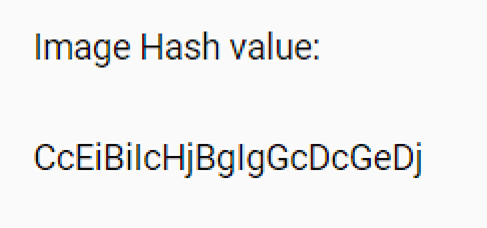


Image Hash value:

CcEiBiIcHjBgIgGcDcGeDj

APPLICATION

Digital Art/NFT verification using image hashing:

Digital art is an artistic work or practice that uses digital technology as a part of the creative or presentation process

NFTs can really be anything digital such as drawings, digital art which uses hashing and blockchain technology to keep track of all the transactions for each trade.

Unique Hash value: GcEhDhCcBdIcAdFiCbFaD

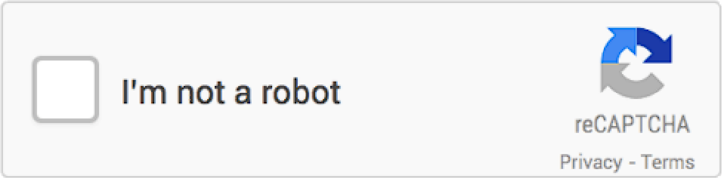
When a buyer buys this unique digital art, it can be confirmed by the buyer that it is the authentic and real art piece by cross checking its hash value and confirming that it is the original digital art.

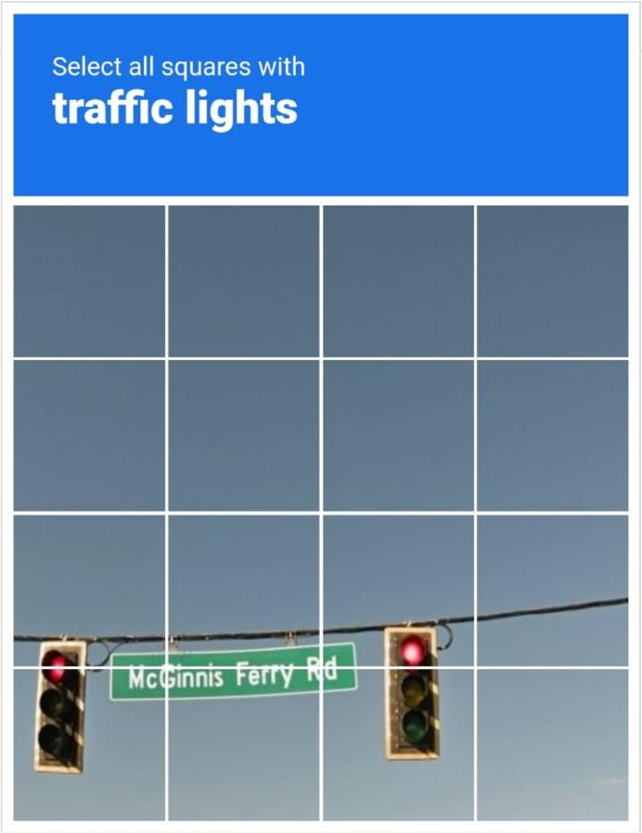
This method can be applied while trading/ selling NFTs (Non-Fungible tokens) to preserve the authenticity by comparing the hash values throughout all transactions

New Captcha method to prevent Malicious Auto-Bots using picture check:

reCAPTCHA protects the sensitive websites from spam and abuse.

Current captchas:

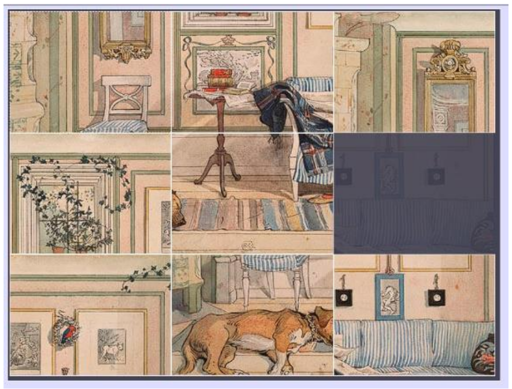



Select all squares with **traffic lights**

This method of prevention is obsolete, and the newer advanced bots can bypass this.

This new method of captcha can make a full proof way of preventing these bots

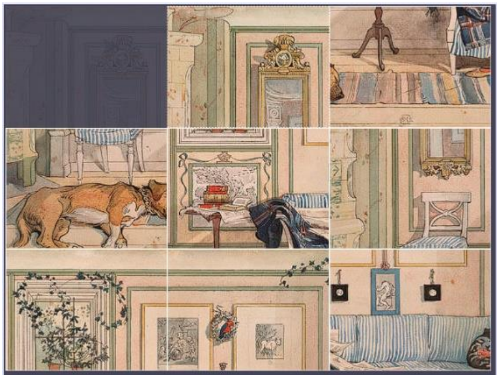
---



Current Hash value : EfFiJhBfAhEcCfDdHfCdIa

Final hash value : FfCdDgHfDfAgHfEaGeAdBa

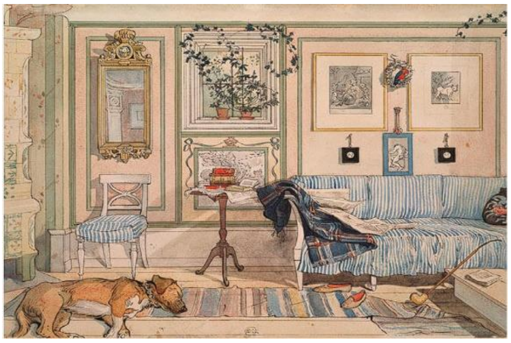
State : not solved



Current Hash value : DfGhJbJgGdFgJfFhJfCdFi

Final hash value : FfCdDgHfDfAgHfEaGeAdBa

State : not solved



Current Hash value : FfCdDgHfDfAgHfEaGeAdBa

Final hash value : FfCdDgHfDfAgHfEaGeAdBa

State : solved

We are comparing the image hash value with its final hash value after which if same, access to a site will be granted

This will be useful as only humans can do this and cannot be easily solved by Bots thus protecting sensitive sites and servers

REFERENCES

1. S. Bhattacharjee and M. Kutter,” Compression tolerant image authentication,” Proc. ICIP-’98, vol. 1, pp. 435-439

2. ROBUST IMAGE HASHING R. Venkatesan1, S.-M. Koon2, M. H. Jakubowski1, and P. Moulin Cryptography Group, Microsoft Research

3. R. Venkatesan and S.-M. Koon,” Robust image hashing into binary strings,” manuscript, 1999