Swap Watermark

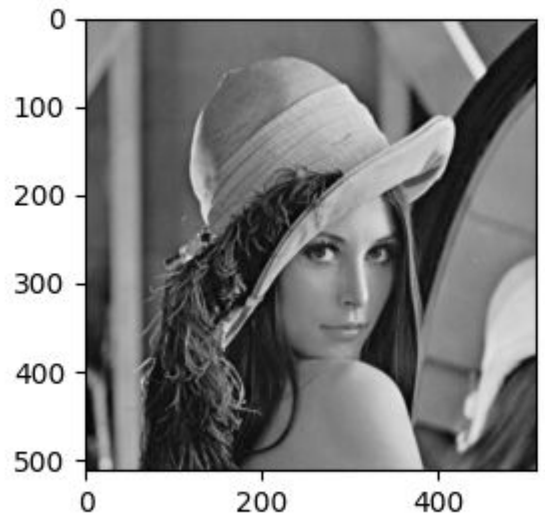
Developers

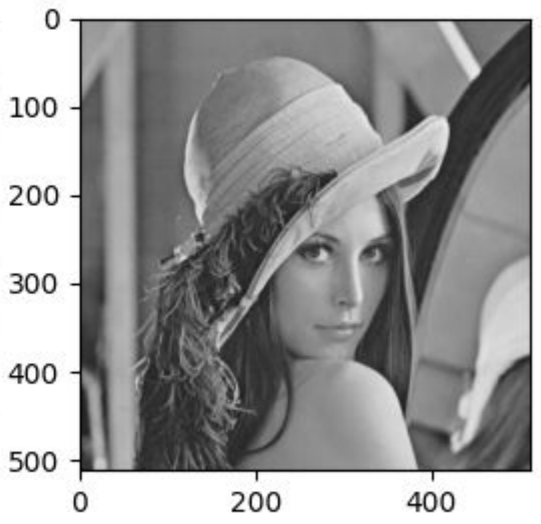
Gabriele Motta
Michele Panariello
Alessandra Morellini
Nicola Arpino

Results (1)

In this part of the presentation, we are proud to show how the algorithm behaves when attacked.

In the following pictures are shown the original image and the watermarked image with a WPSNR of 68.97.

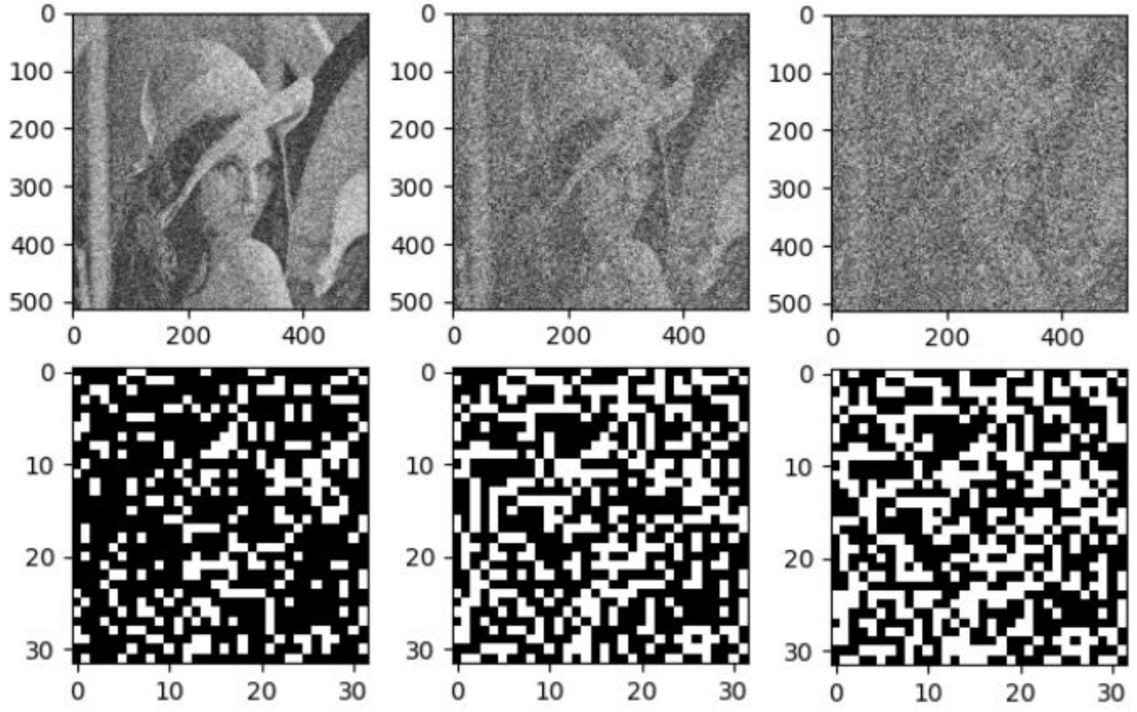




Results (2)

AWGN results: all marks were found with least similarity 12.91:

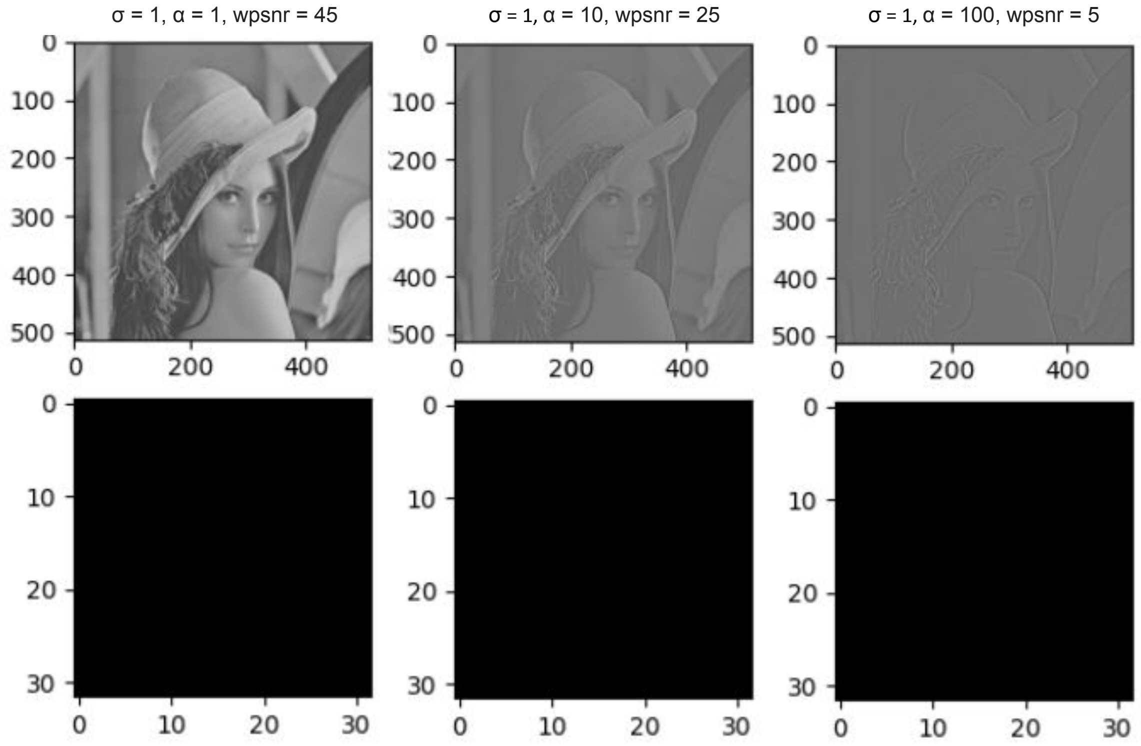
std = 100, wpsnr = 24 | std = 250, wpsnr = 21 | std = 500, wpsnr = 19



ACC 73.0% | ACC 60.0% | ACC 55.5%

Results (3)

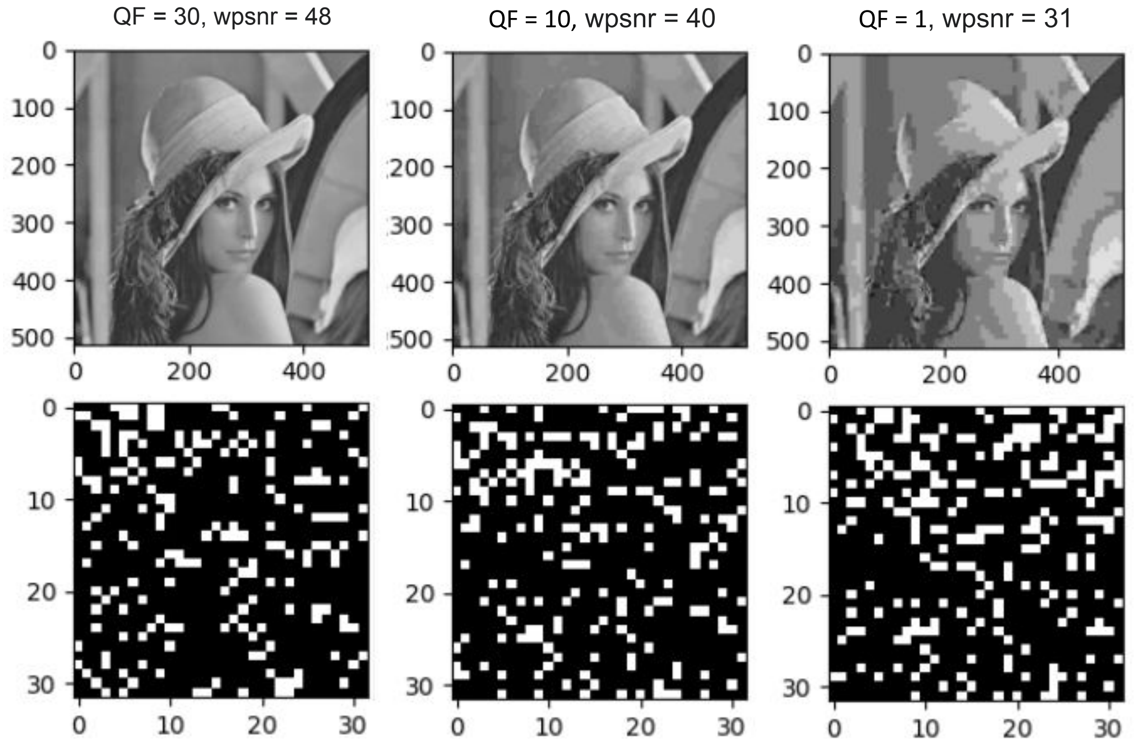
Sharpening results: all marks were found with maximum similarity of 22.69:



ACC 100.0% ACC 100.0% ACC 100.0%

Results (3)

JPEG Compression results: all marks were found with least similarity of 18.69:



ACC 81.5% ACC 80.3% ACC 77.9%

Results (4)

For the sake of completeness we tested our algorithm on 104 given images. The ***average wpsnr is 67.32***

jpeg_compression **qf = 1**

Wpsnr average (a): **31.05**
Accuracy average: **82.19**
Similarity average: **19.44**

awgn **std = 150**

Wpsnr average (a): **21.71**
Accuracy average: **66.48**
Similarity average: **15.12**

resizing **scale = 0.2**

Wpsnr average (a): **31.44**
Accuracy average: **62.27**
Similarity average: **14.06**

blur **sigma = 1.8**

Wpsnr average (a): **35.68**
Accuracy average: **61.67**
Similarity average: **13.55**

median [**3,3**]

Wpsnr average (a): **45.15**
Accuracy average: **73.47**
Similarity average: **16.78**

median [**3,5**]

Wpsnr average (a): **40.56**
Accuracy average: **62.97**
Similarity average: **14.30**

median [**5,3**]

Wpsnr average (a): **40.66**
Accuracy average: **62.98**
Similarity average: **14.38**

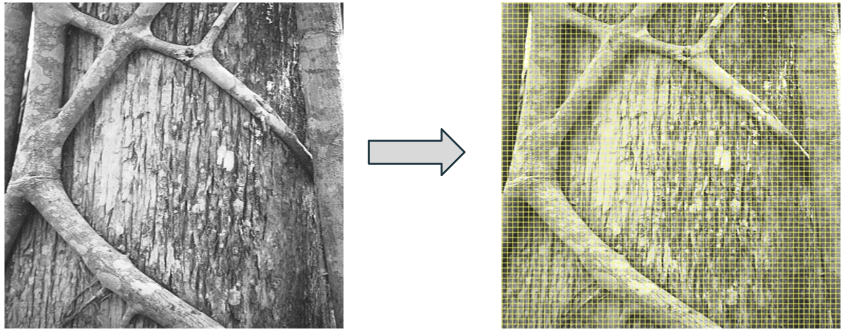
median [**5,5**]

Wpsnr average (a): **38.10**
Accuracy average: **71.48**
Similarity average: **16.22**

Embedding workflow (1)

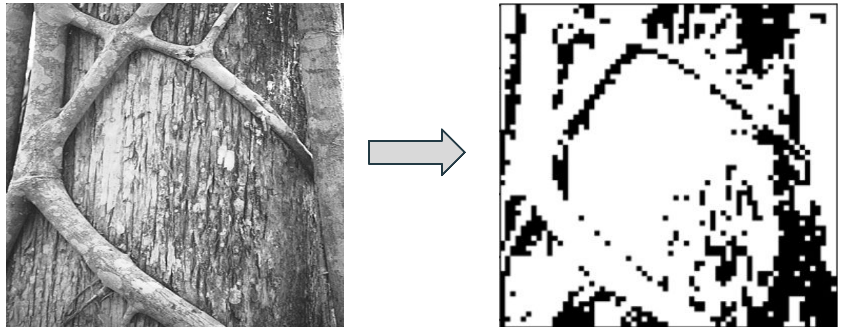
1. First step

Divide the original image into 4096 blocks of 64 pixels each



2. Second step

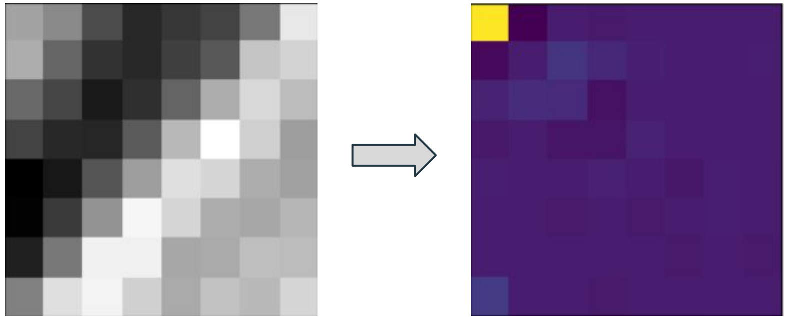
Calculate the ***average*** pixel value for each block and sort them in ascending order. Choose the first 1024 blocks with average > ***optimal average***.



Embedding workflow (2)

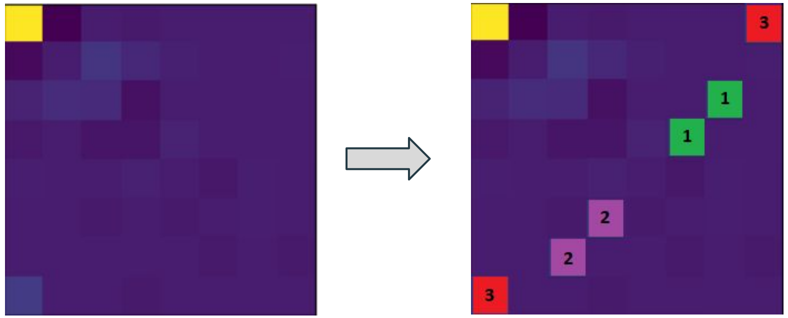
3. Third step

Perform DCT on each i-th block for i from 0 to 1023.



4. Fourth step

Choose a pair of DCT coefficients depending on the ***partitioning*** rule. Swap the coefficients if the i-th mark entry is 1. Boost the value of the greatest coefficient in the pair.

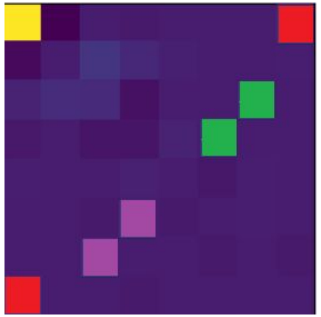


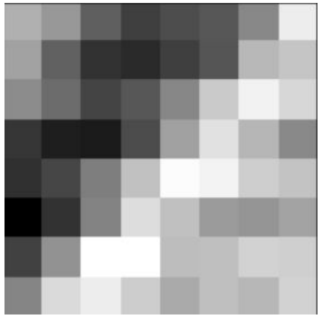
Partitioning rule:
We divide the mark in 3 subarrays. When we embed the first subarray we use the pair of coefficients of indexes [(2, 6), (3, 5)], for the second one we use [(5, 3), (6, 2)] and finally we use the pair [(0, 7), (7, 0)]

Embedding workflow (3)

5. Fifth step

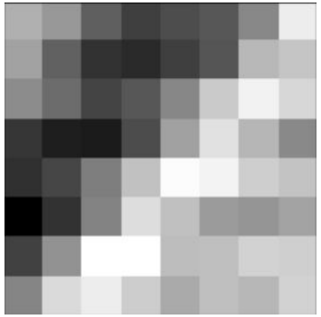
Perform inverse DCT on each i-th modified block of DCT coefficients.





6. Sixth step

Clip each i-th block and get back to the original size



Discussion on parameters

Block size: the block size is very important, because it estimates how much information is embedded in the image for each bit of the mark. We chose 8x8 for the best compromise between quality and robustness.

Optimal Average: we noticed that the np.clip function destroyed the quality of the images almost completely white or almost completely black, or else images many pixels of value 0 or 255. Hence we chose to modify only blocks with average greater than 64, with this parameter the quality of the image is preserved.

Discussion on partitioning

We noticed that the effectiveness of some of the attacks depend on which DCT coefficients are chosen to be swapped and boosted.

Therefore, instead of using only one pair of coefficients, we divided the mark in three parts and embedded each part with a different pair of coefficients.

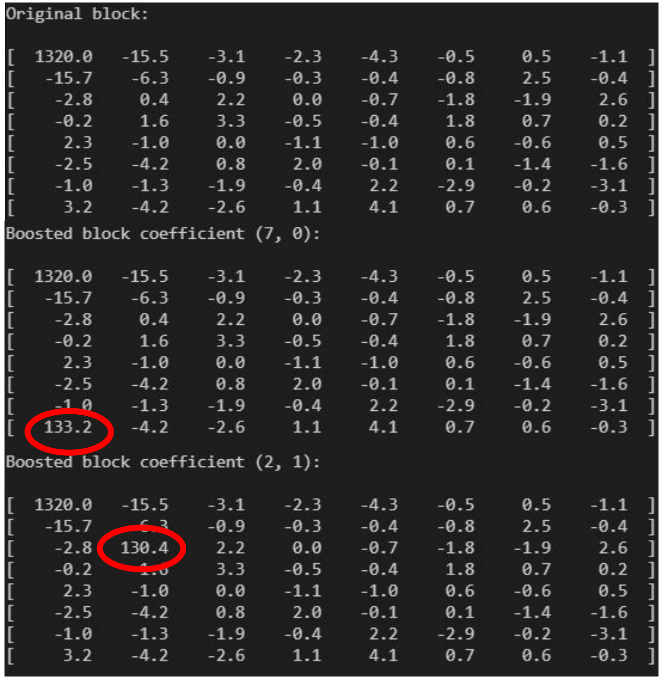
In particular this was caused by median and blur attacks.

More precisely:

1. The pair [**(5, 3), (6, 2)**] resists to the median [a, b] with **a > b.**
2. The pair [**(2, 6), (3, 5)**] resists to the median [a, b] with **a < b**.
3. The pair [**(0, 7), (7, 0)**] resists to the median [a, b] with **a = b**.

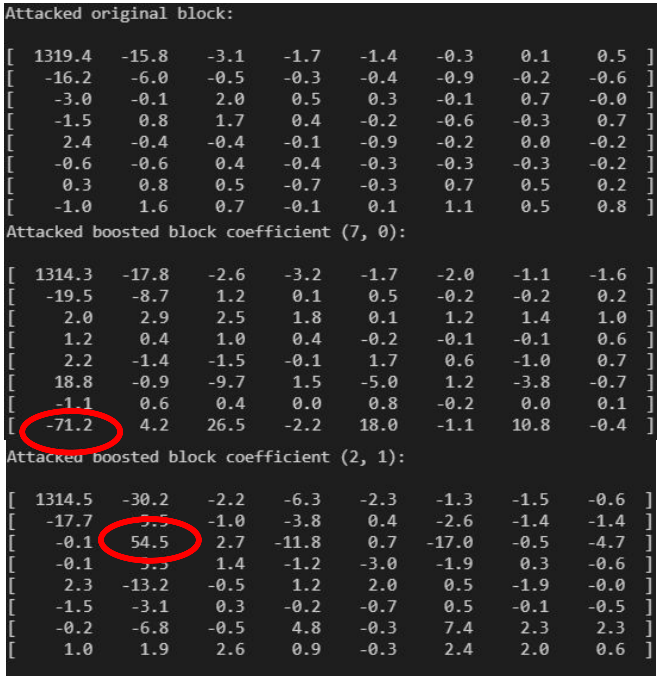
Different coefficients behave differently

We take an image and choose an 8x8 block. One time we boost the coefficient of index (7,0) and the other time the one of index (2,1). We apply a median filtering on the image and notice that the boosted DCT coefficients are significantly increased in absolute value, but the changed its sign in the first case.



```
Original block:

[  1320.0   -15.5    -3.1    -2.3    -4.3    -0.5     0.5    -1.1  ]
[   -15.7    -6.3    -0.9    -0.3    -0.4    -0.8     2.5    -0.4  ]
[    -2.8     0.4     2.2     0.0    -0.7    -1.8    -1.9     2.6  ]
[    -0.2     1.6     3.3    -0.5    -0.4     1.8     0.7     0.2  ]
[     2.3    -1.0     0.0    -1.1    -1.0     0.6    -0.6     0.5  ]
[    -2.5    -4.2     0.8     2.0    -0.1     0.1    -1.4    -1.6  ]
[    -1.0    -1.3    -1.9    -0.4     2.2    -2.9    -0.2    -3.1  ]
[     3.2    -4.2    -2.6     1.1     4.1     0.7     0.6    -0.3  ]
Boosted block coefficient (7, 0):

[  1320.0   -15.5    -3.1    -2.3    -4.3    -0.5     0.5    -1.1  ]
[   -15.7    -6.3    -0.9    -0.3    -0.4    -0.8     2.5    -0.4  ]
[    -2.8     0.4     2.2     0.0    -0.7    -1.8    -1.9     2.6  ]
[    -0.2     1.6     3.3    -0.5    -0.4     1.8     0.7     0.2  ]
[     2.3    -1.0     0.0    -1.1    -1.0     0.6    -0.6     0.5  ]
[    -2.5    -4.2     0.8     2.0    -0.1     0.1    -1.4    -1.6  ]
[    -1.0    -1.3    -1.9    -0.4     2.2    -2.9    -0.2    -3.1  ]
[   133.2    -4.2    -2.6     1.1     4.1     0.7     0.6    -0.3  ]
Boosted block coefficient (2, 1):

[  1320.0   -15.5    -3.1    -2.3    -4.3    -0.5     0.5    -1.1  ]
[   -15.7    -6.3    -0.9    -0.3    -0.4    -0.8     2.5    -0.4  ]
[    -2.8   130.4     2.2     0.0    -0.7    -1.8    -1.9     2.6  ]
[    -0.2     1.6     3.3    -0.5    -0.4     1.8     0.7     0.2  ]
[     2.3    -1.0     0.0    -1.1    -1.0     0.6    -0.6     0.5  ]
[    -2.5    -4.2     0.8     2.0    -0.1     0.1    -1.4    -1.6  ]
[    -1.0    -1.3    -1.9    -0.4     2.2    -2.9    -0.2    -3.1  ]
[     3.2    -4.2    -2.6     1.1     4.1     0.7     0.6    -0.3  ]
```

median [3,3]

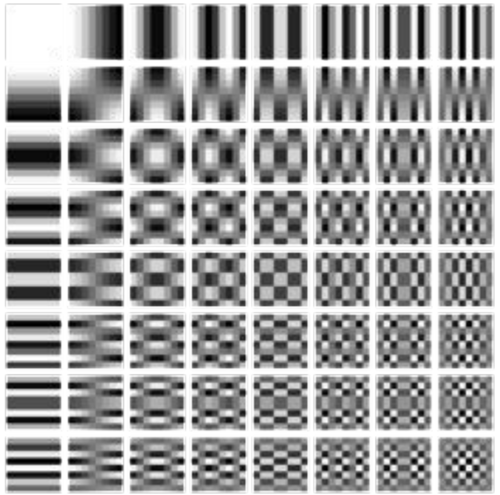


```
Attacked original block:

[  1319.4   -15.8    -3.1    -1.7    -1.4    -0.3     0.1     0.5  ]
[   -16.2    -6.0    -0.5    -0.3    -0.4    -0.9    -0.2    -0.6  ]
[    -3.0    -0.1     2.0     0.5     0.3    -0.1     0.7    -0.0  ]
[    -1.5     0.8     1.7     0.4    -0.2    -0.6    -0.3     0.7  ]
[     2.4    -0.4    -0.4    -0.1    -0.9    -0.2     0.0    -0.2  ]
[    -0.6    -0.6     0.4    -0.4    -0.3    -0.3    -0.3    -0.2  ]
[     0.3     0.8     0.5    -0.7    -0.3     0.7     0.5     0.2  ]
[    -1.0     1.6     0.7    -0.1     0.1     1.1     0.5     0.8  ]
Attacked boosted block coefficient (7, 0):

[  1314.3   -17.8    -2.6    -3.2    -1.7    -2.0    -1.1    -1.6  ]
[   -19.5    -8.7     1.2     0.1     0.5    -0.2    -0.2     0.2  ]
[     2.0     2.9     2.5     1.8     0.1     1.2     1.4     1.0  ]
[     1.2     0.4     1.0     0.4    -0.2    -0.1    -0.1     0.6  ]
[     2.2    -1.4    -1.5    -0.1     1.7     0.6    -1.0     0.7  ]
[    18.8    -0.9    -9.7     1.5    -5.0     1.2    -3.8    -0.7  ]
[    -1.1     0.6     0.4     0.0     0.8    -0.2     0.0     0.1  ]
[   -71.2     4.2    26.5    -2.2    18.0    -1.1    10.8    -0.4  ]
Attacked boosted block coefficient (2, 1):

[  1314.5   -30.2    -2.2    -6.3    -2.3    -1.3    -1.5    -0.6  ]
[   -17.7    -5.5    -1.0    -3.8     0.4    -2.6    -1.4    -1.4  ]
[    -0.1    54.5     2.7   -11.8     0.7   -17.0    -0.5    -4.7  ]
[    -0.1     3.3     1.4    -1.2    -3.0    -1.9     0.3    -0.6  ]
[     2.3   -13.2    -0.5     1.2     2.0     0.5    -1.9    -0.0  ]
[    -1.5    -3.1     0.3    -0.2    -0.7     0.5    -0.1    -0.5  ]
[    -0.2    -6.8    -0.5     4.8    -0.3     7.4     2.3     2.3  ]
[     1.0     1.9     2.6     0.9    -0.3     2.4     2.0     0.6  ]
```

Detection

Main characteristics

- Works in DCT domain
- Block based (8x8) -> 4096 blocks
- Dynamic detection considering blocks average value
- Dynamic spots check



Detection workflow

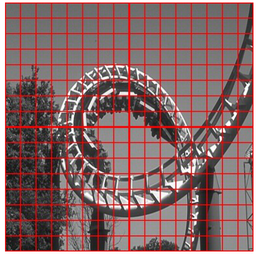
original mark extraction 1/2

1. First step

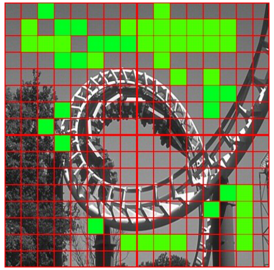
Divide original image into 4096 blocks. Each block is 8x8 pixels.

2. Second step

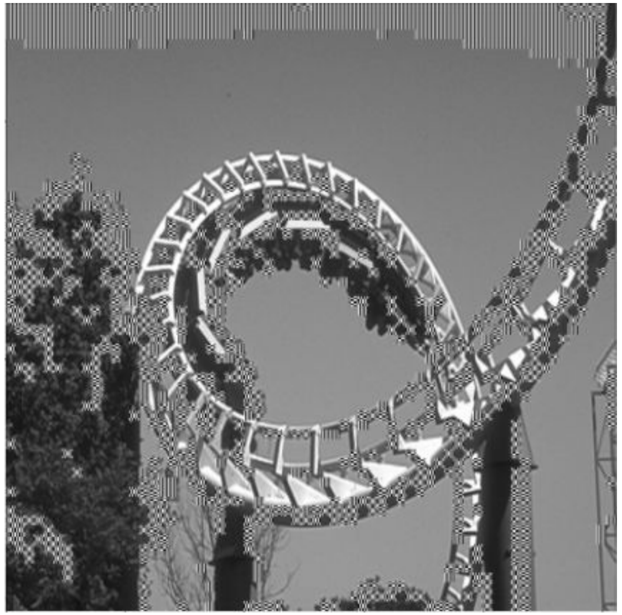
Calculate the ***average*** (mean value) for each block and sort them in ascending order. 1024 are then used.



4096 chunks division



Example of 1024 possibile blocks sorted



Actual 1024 blocks considered

Detection workflow

original mark extraction 2/2

3. Third step

Perform DCT on watermarked and original image on each block.

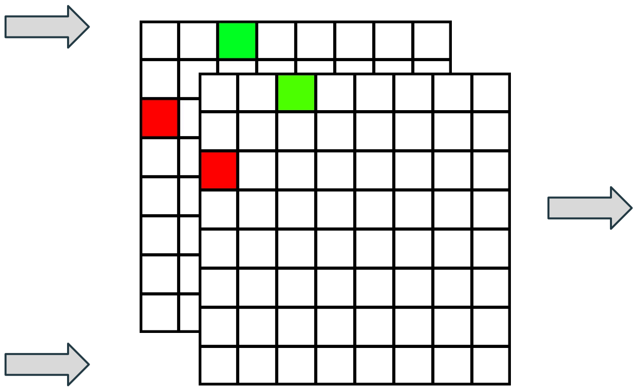
4. Fourth step

Compare DCT blocks spots following sort order and ***partitioning*** rule.

Watermarked (mark is enhanced)

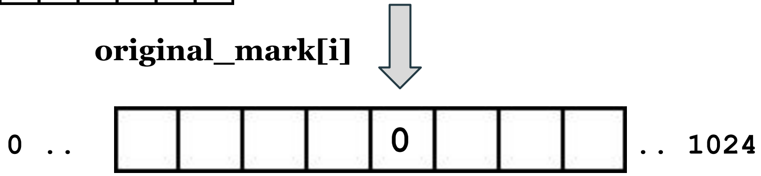
Original

Comparison of DCT coefficient at given spots (2,0)(0,2)



Extraction rule pseudocode

```
If the green spot
of the original is
> or < than the
red one AND this
relationship holds
also in the
watermarked then
extract ZERO
```



Detection workflow

attacked mark extraction 1/2

Extraction steps

1. Same as original mark extraction.

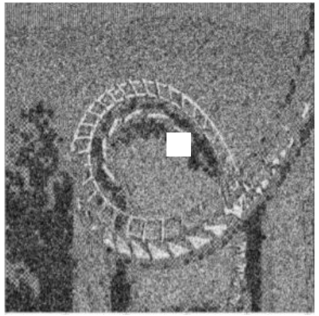
2. Perform **accuracy** calculation:

Bitwise comparison between original mark and attacked one.

Formula:

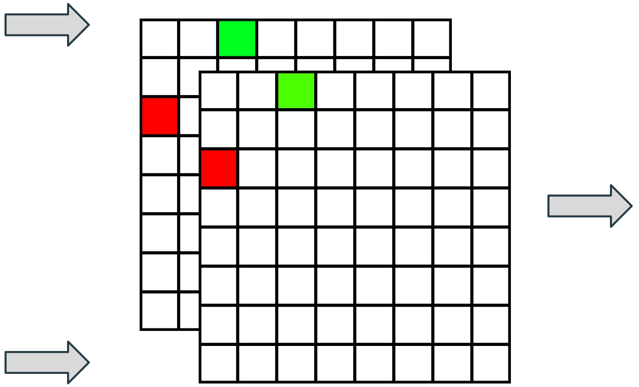
(Same bits / 1024)*100

Attacked



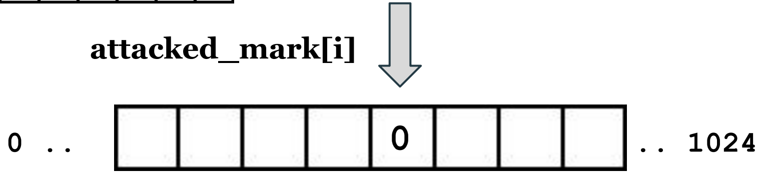
Original

Comparison of DCT coefficient at given spots (2,0)(0,2)



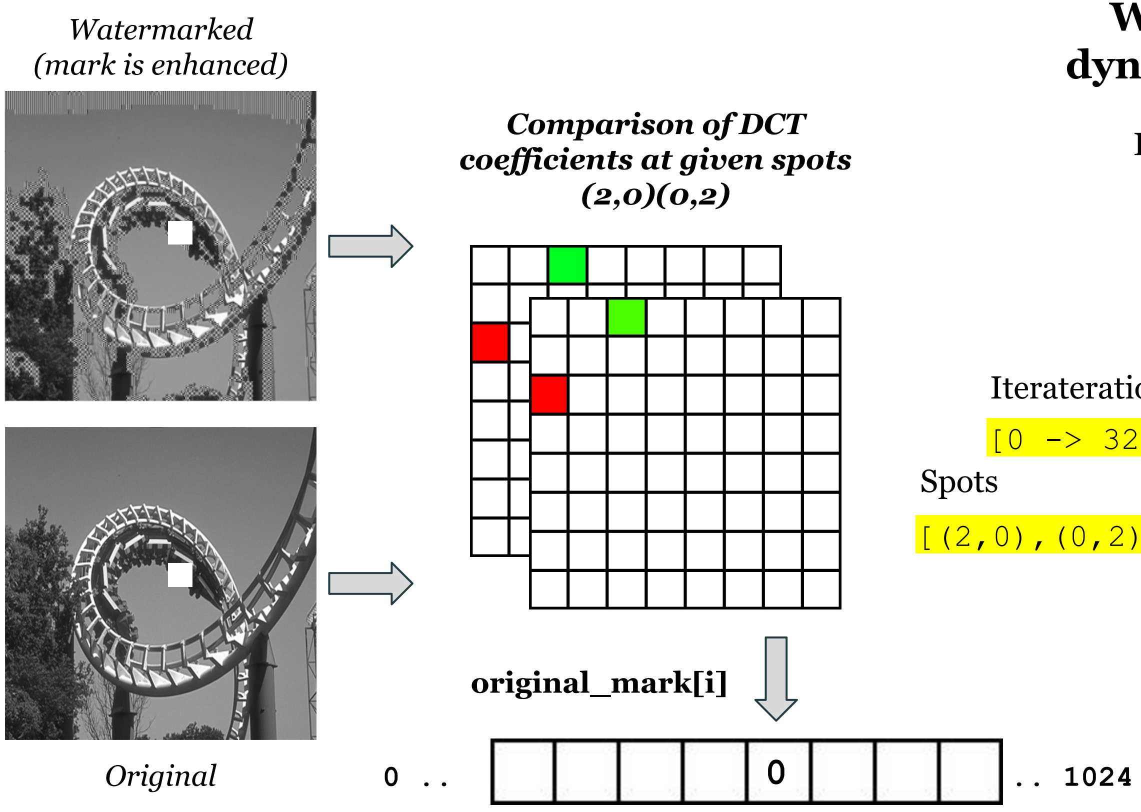
Extraction rule pseudocode

```
If the green spot
of the original is
> or < than the
red one AND this
relationship holds
also in the
watermarked then
extract ZERO
```



Detection workflow

original mark extraction, dynamic spots selection



What is intended for dynamic spots selection?

IDEA: Based on iteration “i” a different pair of spots is selected according to ***partitioning***

Iterateration ranges

`[0 -> 325]` `[325 -> 650]` `[650 ->1024]`

Spots

`[(2,0),(0,2)]` `[(0,7),(7,0)]` `[(5,3), (3,5)]`

Detection workflow

attacked mark extraction 2/2

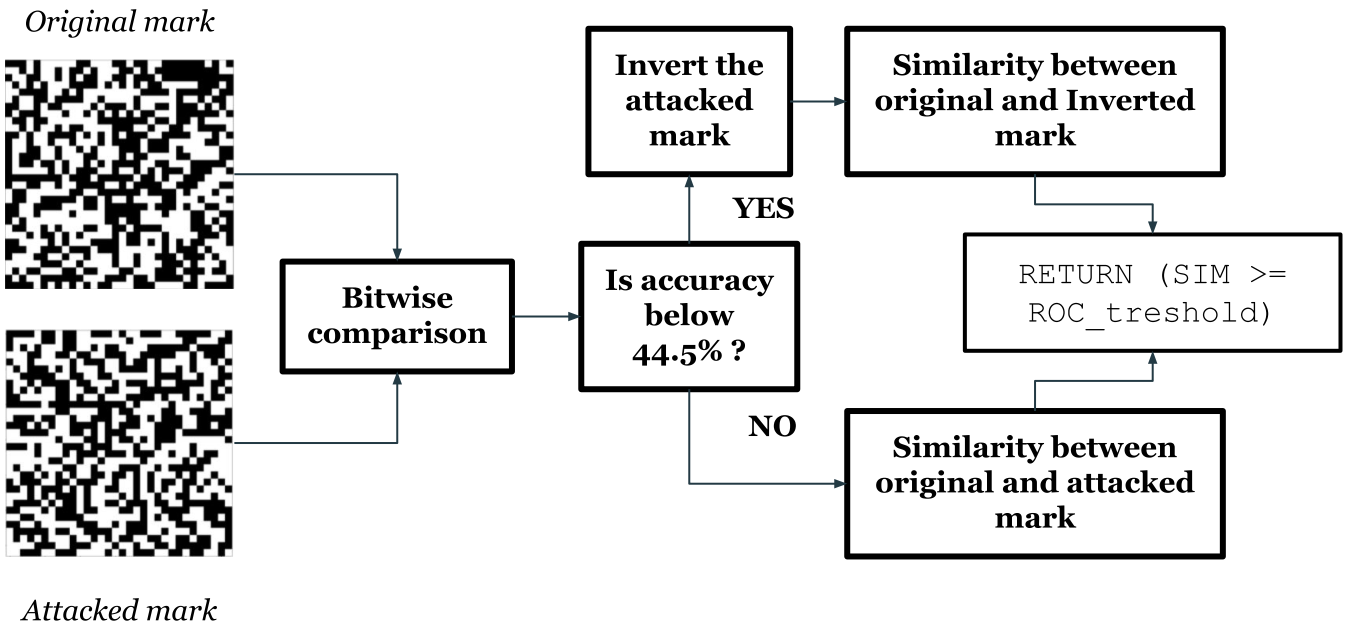
Is the mark still there?

1. Compute accuracy

2. Check if accuracy is below the ***inverting limit.***

Similarity is then computed on original mark and the attacked one which can be in inverted in some cases.

3. Return SIM comparison and WPSNR



Original mark
Attacked mark
Bitwise comparison
Is accuracy below 44.5% ?
YES
NO
Invert the attacked mark
Similarity between original and Inverted mark
Similarity between original and attacked mark
RETURN (SIM >= ROC_treshold)

Detection workflow

Mark inversion example

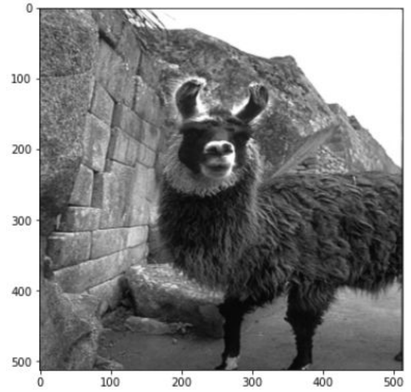
Median asymmetric attacks such as Median(3,5) makes our mark inverted!

Settings

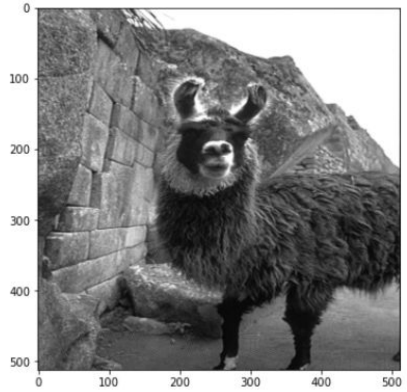
Median(3,5) attack

Wpsnr (watermarked / original): 69.2
Wpsnr (attacked / watermarked): 43.9
Accuracy = **33.0078125%**
Mark has been lost. Sim: 6.84
Threshold: 12.18

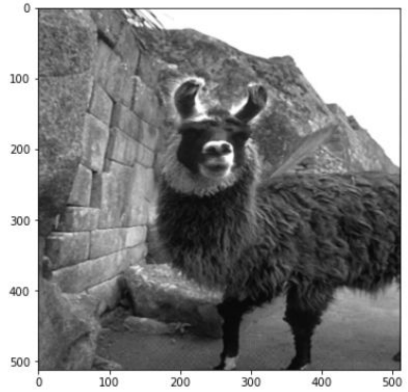
Original

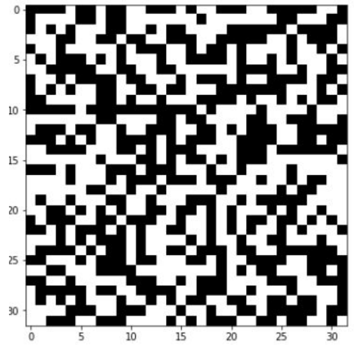


Watermarked

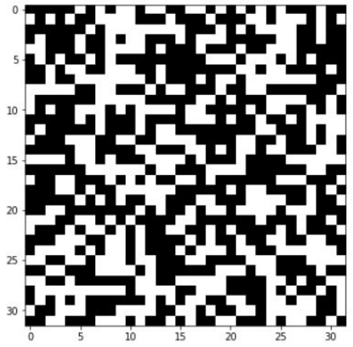


Attacked

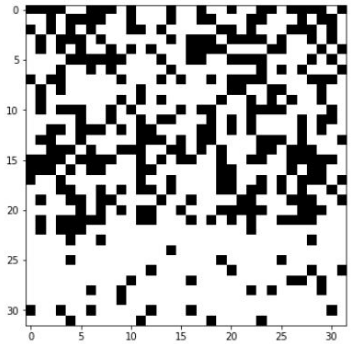




Original mark



Attacked mark



Bitwise comparison

Detection workflow
Mark inversion FIX

Our solution is to check for inverted mark.

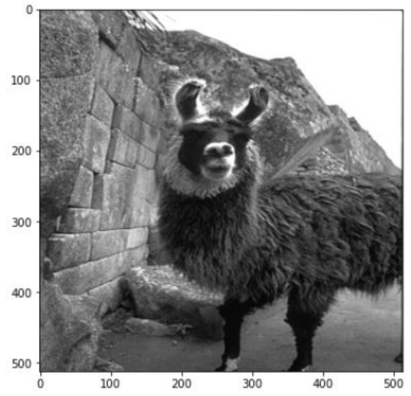
Inverting mark threshold 44.5%

Settings:

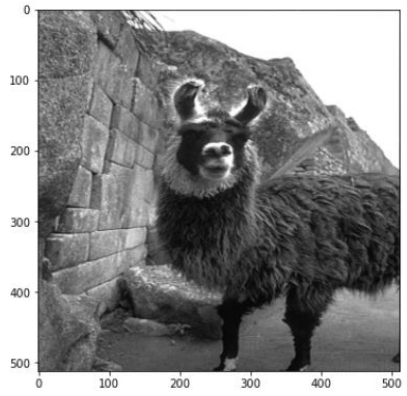
Median(3,5) attack

Wpsnr watermarked = 69.2
Wpsnr attacked: 43.9
Accuracy = **66.9921875%**
Mark has been found. Sim: 15.55
Threshold: 12.18

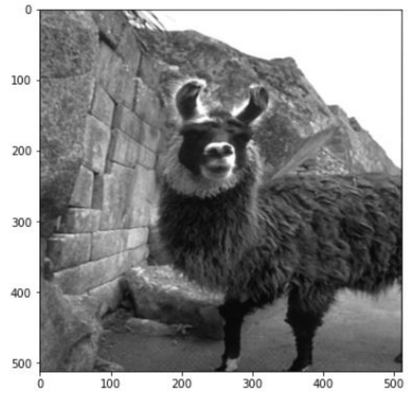
Original

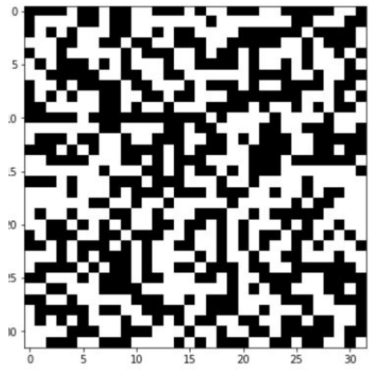


Watermarked

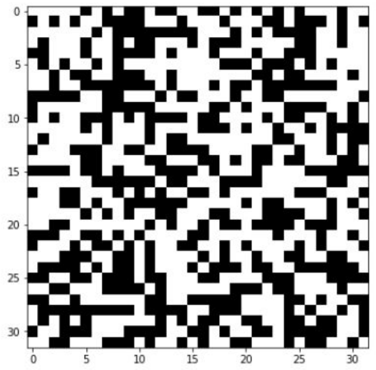


Attacked

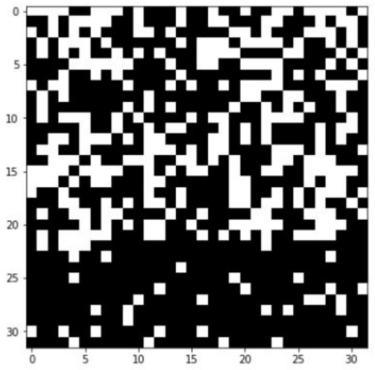




Original mark



Attacked mark



Bitwise comparison

Detection workflow
Execution example

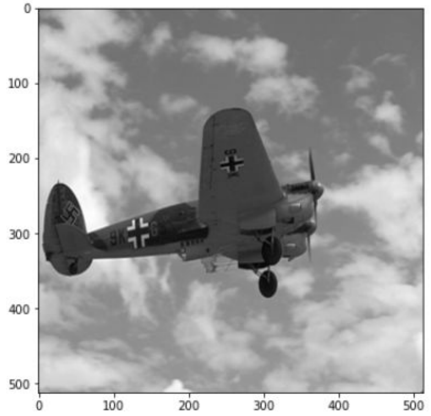
Settings

Attack AWGN(100)

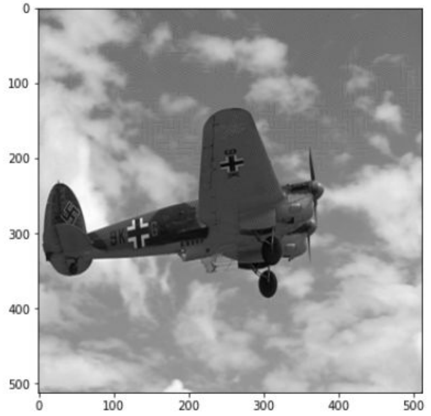
Wpsnr watermarked = 65.5
Wpsnr attacked = 24.2
Accuracy = **72**%
Similarity = 16.44

Threshold 12.18

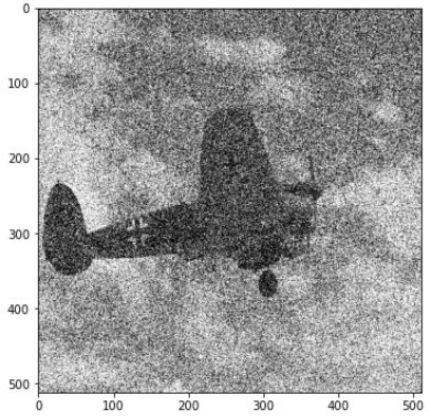
Original

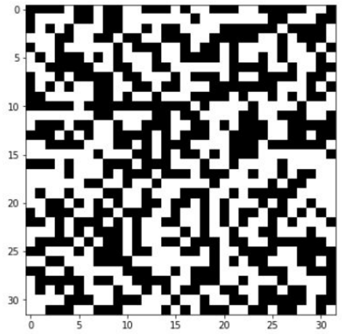


Watermarked

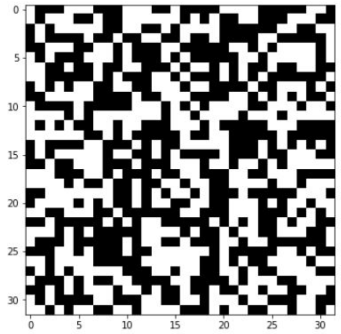


Attacked

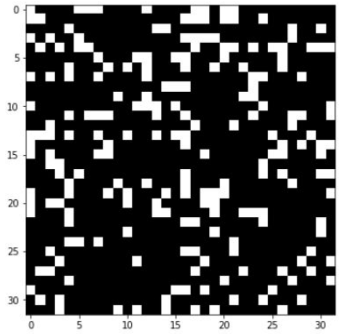




Original mark



Attacked mark



Bitwise comparison

Detection workflow
Further improvement

Asymmetric Blur block wise cause portions of the mark to invert in sign independently from other portions

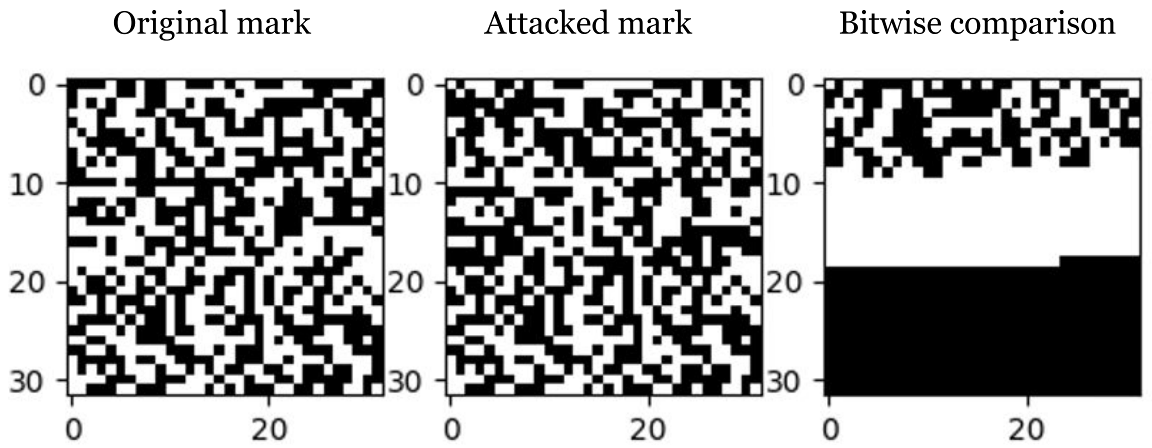
Inverting mark threshold 44.5%

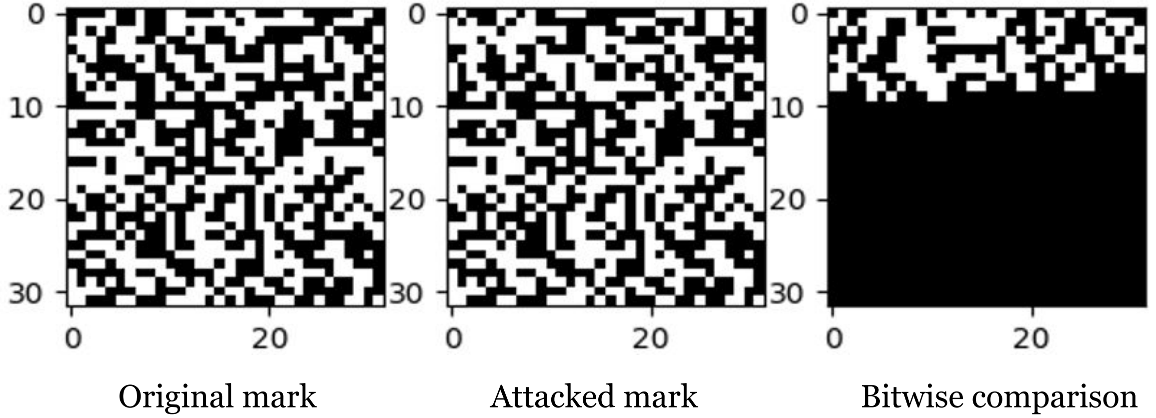
Settings:

Blur(2, 2.5) block wise

Accuracy ~ **53%**
Mark not found

Accuracy after fix ~ **84%**
Mark found


Original mark
Attacked mark
Bitwise comparison


Original mark
Attacked mark
Bitwise comparison