Using Mean Shift Segmentation for Key Space using a Flavor of the Block Chain Cipher

Author: *Madeleine Sirok*
Computer Science Dept.
Florida Polytechnic University
Lakeland, United States of America

***Abstract*— This is a novel implementation of combining block chain and other encryption techniques by using image segmentation for a key space. Mean shift segmentation is used to create a sized key space based on the number of segments created within the image. The mean of all pixels in a segment is calculated to create a red, blue, and green code which is converted to hex code to become a six-character block. The inputted plaintext is blocked into six-character blocks to match the size of the hex code to be encrypted using the block chain technique. The ASCII code for each hex character and plaintext character in a block is encrypted and creates a block of ciphertext. The outcome of the process generates a ciphertext that can only be decrypted with the exact image used for encryption else the generated image key space is different.**

Keywords—block chain encryption, block chain, encryption, image segmentation, mean shift segmentation

I. Introduction

In this paper, a novel implementation of text encryption using image segmentation will be examined. The image will be divided into segments using a mean shift algorithm. Those segments of pixels will be placed into a bin in which their red, green, and blue (RGB) values are averaged to find the mean of the segment. Those mean values will be converted from RGB to hexadecimal (hex) value. That hex value will consist of six characters which will form the key block for the encryption process. Each segment will represent a key block within the key-space. The plaintext would be divided into six-character block sizes as well to match with the hex keys. For this paper, a modulo cipher was used for the letter encryption process to exemplify the idea using a less complex symmetric encryption method. This paper will explain what image segmentation is and the process used for encryption, explain what block chain encryption is, and the modifications to the algorithm for this encryption process. The testing process and results of this algorithm will be discussed to prove the level of security for this encryption method.

The environment for implementing this was PyCharm CE IDE on MacOS and coding with the Python language. The different libraries used for this implementation include OpenCV (version 4.5.5.62), sklearn.cluster (version 1.0.2), and numpy (version 1.21.4). Note that downloading each individual library is different within each environment and external research for downloading should be used.

II. Image Segmentation

A. Image Segmentation

Image segmentation is partitioning the image into regions or clusters based on similarity in color. This allows for meaningful segments to be produced such that the sub images have meaning and purpose. Mean shift segmentation, the type used for this encryption algorithm, is a method in which a pixel on the image is seeded and the mean shift algorithm is used to find where the densest region of a color is [1]. A weight is given to the pixels with the color corresponding to the color of interest so that it can put that pixel in a bin for that segment. Depending on how the segmentation is coded, the segments can be colored by preference (hard coded) or by calculating the mean of all the pixels within that segment (method that will be used within this encryption). Figure 1 shows the original image before segmentation and Figure 2 exemplifies what image segmentation looks like for that image. The different shades and colors of blocks are segmented into different bins and output a specified grayscale color. With a lot of contrast within the image, the pixels are more easily separated into bins so that there is less noise within the segments.

Fig. 1. Unsegmented image of toy blocks

Fig. 2. Segmented image of toy blocks

The number of different segments was hardcoded for the output given in Figure 2, but with the given mean shift function using the sklearn cluster library in python, the estimated bandwidth parameters can be edited so that a different number of segments can be outputted. The bandwidth function is stated as:

$$\begin{aligned} &sklearn.cluster.estimate\_bandwidth(\\ &\quad X: Any, *, \\ &\quad quantile: Any = 0.3, \\ &\quad n\_samples: Any = None, \\ &\quad random\_state: Any = 0, \\ &\quad n\_jobs: Any = None) \end{aligned} \tag{1}$$

Where X is the image being used and the quantile is the median of all pairwise distances [4]. The number of samples/seeds the algorithm runs through to find the best fitting segment for that section, the pixels could be placed into different bins.

This is the process that is used for experimenting and testing this novel encryption method. It was used in a previous project where [1] where mean shift segmentation and Otsu binarization methods were implemented on images. The code and methodization for this cipher was expanded upon to get different assets from the image segmentation.

III. Block cipher encryption/decryption

B. Block Ciphers

The block cipher is a method of encryption in which blocks of text are encrypted using a simple operation and a set of keys. This method uses bit-level encryption and the exclusive-or (XOR) function to produce ciphertext [7]. Since the produced blocks of text have to be looped through to reach the end of the plaintext, there must be a series of 'rounds' for the *n* number of blocks. Note that the same encryption method is used for each round.

There are a multiple of block cipher encryptions that can use the ciphertext output of the (*n-1)*th round as the *n*th round key or produce ciphertext blocks that concatenate on the last round. The linking of ciphertext and the next round of encryption depending on the previous round causes a link – hence naming it block chain encryption.

The encryption method within this paper uses a flavor of block chain encryption that directly encrypts each block with a key in the key space and produces a cipher block. It also does not use bit-level encryption, rather it uses the ASCII code of each character and uses modulus to produce ciphertext. Table I defines the encryption and decryption algorithm used to encrypt/decrypt each character within the block of text.

TABLE I. Block Cipher Encryption Algorithm

Encryption	***Decryption***
ASCII(a) + ASCII(b) mod 128	*ASCII(a) - ASCII(b) mod 128*

IV. Implementation and testing

The implementation of this method was writtin in Python using the OpenCV, sklearn.cluster, and numpy packages. This section will discuss the different components used for the text encryption such as the images used and their respective key space, the plaintext, and the ciphertext (encrypted with the correct key image).

The same plaintext was used for all the encryptions. For sufficient testing, the plaintext should have more than 50 characters. This will allow an image to provide at least 7 keys in the key space and loop through the key space partially.

C. Method Implementation

To implement this method, multiple different photos were chosen to be used for the key space. The photos used include I_1, I_2, and I_3. I_1 is the true image key for encrypting the text. I_2 is a slightly altered version of I_1 in that its hue was shifted to be +11 more points within Photoshop. I_3 is a completely different image to show that only a single image will be able to decrypt the ciphertext. I_4 is an image with similar hues and colors as I_1 to show that similar images won't decrypt the ciphertext properly.

Fig. 3. Depicts I_1: True Image Key

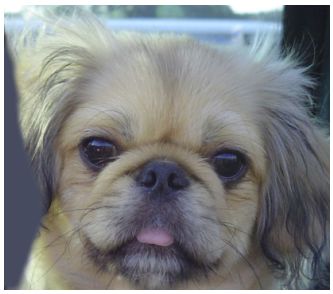

Fig. 4. Depicts I_2: Altered Version of I_1 with +11 Hue

Fig. 5. Depicts I_3: Different Image than I_1 and I_2

Fig. 6. Depicts I_4: Image with near hues of I_1

Each of these images were segmented using mean shift to produce a key space that will be used for both encryption and decryption of the plaintext. The table below shows the output of hexadecimal values produced by each respective image.

TABLE II. KEY SPACE PRODUCED BY EACH SEGMENTED IMAGE

Image Name	*Number of Segments*	*Key Space (Hex Values of Segments*
I_1	7	'595251', '7b7e80', '9ea39e', '322b28', 'c1ccbf', 'f1e2ba', 'f9fbf5'
I_2	7	'595051', '7b7e7f', '9ea39e', '312928', 'c1cbbe', 'f0d8ba', 'f9faf5'
I_3	17	'808689', '576266', '4ecacb', '1f22b0', '489953', '13145d', '583414', '252018', '293a45', 'b68143', '775028', '689ba3', '5cb3b9', '375473', '8b785f', '81633f', '4c6bae'
I_4	6	'657994', '8ca6c0', '514f3f', '3a393c', '23293e', '010101'

As can be seen from Table II, I_1 and I_2 have a few of the same key blocks which does not allow for sufficient security if there are only minor changes to color of an image. The image key requires unique values and 'random' enough where it is semantically secure such that another random image has a low probability of containing a segment with the same color as the key image. This type of security can be seen if an attacker attempted to use an image like I_3 where there are over ten more segments generated and none of the segments being similar to I_1. Though I_4 has similar hues and colors as I_1 the number of segments produced, and the key space is still very different from I_1.

The plaintext used throughout this process is a famous merged set of dialogue from Star Wars: Episode III. The quote is

"*Did you ever hear the tragedy of Darth Plagueis The Wise? I thought not. It's not a story the Jedi would tell you. It's a Sith legend. Darth Plagueis was a Dark Lord of the Sith, so powerful and so wise he could use the Force to influence the midichlorians to create life… He had such a knowledge of the dark side that he could even keep the ones he cared about from dying. The dark side of the Force is a pathway to many abilities some consider to be unnatural. He became so powerful… the only thing he was afraid of was losing his power, which eventually, of course, he did. Unfortunately, he taught his apprentice everything he knew, then his apprentice killed him in his sleep. Ironic. He could save others from death, but not himself."* – Chancellor Palpatine [5]

The next component in the process is finding the ciphertext corresponding to the true key image – I_1 – which will be attempted to be decrypted using the key space provided by I_2 and I_3.

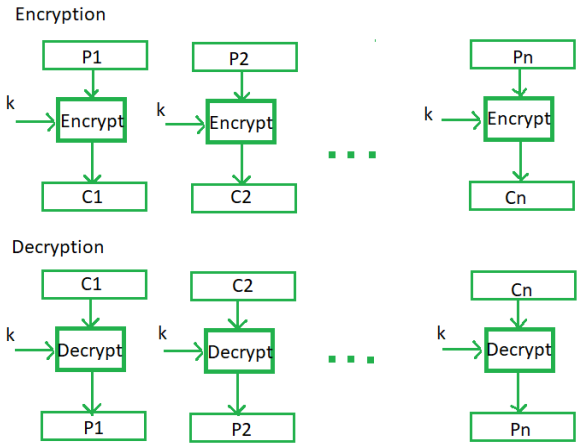

Fig. 7. Block Chain Cipher Algorithm

Figure 6 provides a visualization of how the block chain cipher works [6]. A block of plaintext is encrypted with a block of key to create a block of ciphertext. All the ciphertext blocks are concatenated to generate the full ciphertext as a single string. To decrypt, it is the same process except reversed; a block of ciphertext is decrypted with a block of key (same block used to encrypt) to generate a block of plaintext. If done correctly, the concatenated plaintext is the same as the original plaintext.

TABLE III. CIPHERTEXT GENERATED BY I_1

Ciphertext using Key Space from I_1
y¢□R® ¬□□Ù□¢YÍÆ□«□§□□□¦ªÄ□ÈÇÙ□Õ□□vÃÓÚ¡□²Ò□□®□ □¨Q□Ê□□□□¬Ê S□□§□¡×□ ×QÑÒÖ□□zÙ‖Õ□Ô¨Ú□ÇU¨-¤¤®Q«Ê □□□□□Î□ª¨Ú□□RÖ□¤ÏQÜÒ×□□zÙ‖Õ□ÇY¹ËÚ□U¥□□□□□□W© □¢-Í□□¥Æ□§□Ë¥XÚ□Ö□Ã□ª□×□□-Õ«Ê□Õ□U-□□U□ Ö□□X£¨□

Ñ¢°Ê¥□§ÎR□Ñ□□ÖÑ□Ý□Ø□□ÊËÝÉÑÛ¡□Yª¥□Q«È□□~□«ÈÆS-Ô
S□ È□-È□ÆÈ□ÚÎ□□□ËÅÏ□ÎÎÕ§□□£¥U¥¦□□×□□-È□□¢Ë□÷·Rª□X
Ë□Ç□ÕÛÉ□□□□ÌÔ¨ÝÎË□□□U¡□Q«Ê□□□□«Ð□¦¢É□R¦Ê□¬□□È□
ÅÕÛ□ÉRÇ×Ë§□ÍË□¥Y©□□Q¦Ð□ØX□□□Ä□«Ê□R□Ä¡-×QÉÕÑÓ□
□Þ□ÐÈ□Yº□ÊÛ□□§□U¤ Æ□□§□YÙÉ□Y«¢¤□ÇR¡ÖQÄ□ÒÇÚ□Ü□
Û□Ú¨□ÏÇ£®Y□□□□ Ö Ê«P¬ÔÎ□YÈ¢ ¥Ë□□ÕQ×Ò□ÈÊQÚ ÐÂÚ®Ø
ÃÒcU□□R□□□Ã¤ÊX£¨□Ñ¢°Ê¥□§Î·X×□È□ÑÔÒª□¦ÊÊÔ □ÊËU¬□¨
R□□©Ã ÉX□□□Ø□¬□□¡¥Ë □□□ÌÖ□ÖÕ¨Ê¤□□Ý¡ÏÅÎU□¯□ ©¦□Î£
ÞdP¨Ë□□¨Ú¥¥□□R ÈQÇÌÆ□□□Ó□ÑÓÚ®ÔÃÚ□¡²aR□□WÖ□Ú□□-
□É□¬□□¢¢Ô□¦×□ÆÈ□ËÜ□×«ÖÉÏ§Í□Î□U¤£□¬]WÖ□Ê¦P¡ÎÔS□Õ£¤
□Ð¦¡Æ□□ÎËÒÒ□ÉRÊÊÓYÏÐ□□□¬U¥¡□□Òe□□¢¨ÓÊ□g□{□RÅ¡-Ï□
□ÖÃÜËQÒ¦ÊÆØ¬□ÈØ¤¢Y□□□¥□□WÇ-¤YÓÐ§YÍ□□¥Ç□□□a□□
□□

The ciphertext generated by using the plaintext and the key space from I_1 can be seen in Table III. The '□' symbol represents a blank space.

D. Testing the Code

The testing phase was completed through modular testing – making sure each function works properly before testing them together. The four different modules within this implementation are mean shift segmentation, conversion functions for blocking and hex/RGB numeric, and block chain encryption, and block chain decryption.

Once the mean shift algorithm worked as indented, different images were fed through to compare the different outputs each image produced. The two key features that are looked at are the number of segments and the mean color of each segment. These two features and their variance will define how easy or difficult it is to brute force decrypting the ciphertext.

The helper functions include sectioning strings into blocks of six characters and converting RGB values to hexadecimal [2]. The blocking method was tested with different lengths of strings to be sure that each one correctly chunked the string into blocks of six characters. If the final block does not have all six characters, padding will be added such that six characters are in the block. The RGB to hexadecimal conversion was tested by using an online conversion calculator to be sure that the outputs are correct each iteration.

The block chain encryption and decryption functions were tested modularly but needed each other to be working properly to show that the processes are working. Simple strings were input with simple key blocks to test if the encryption was working properly, and then decrypted to show that the original string was returned. An online encryption calculator was used to double check these results as well.

V. Results and Conclusion

The results of this method are explained through step-by-step implementation of the cipher. The different components needed and defined are the images, their key space, the plaintext, and the attempted plaintext for each respective ciphertext decryption.

First, the user must have the key image and use the exact same segmentation process for all image segmentation, or the results will differ. To obtain ciphertext, the user uploads their image into the same file folder as the code to use direct reference of the image. Once the code is run, all steps of segmentation, key space creation, and encryption will be done as long as the functions are referenced in the main() function. The main() function can be manipulated to segment one or more images and encrypt/decrypt from the images segmented. To test and produce these results, three images were segmented to create their respective key space. The correct image key, I_1, was the only image used to encrypt the text, and all three images I_1, I_2, and I_3, were used to decrypt the ciphertext. From there, an analysis is done to see how similar the produced plaintext is to the true plaintext.

TABLE IV. Comparisons of Encrypted Text with Image Changes

Image Key Space	***Plaintext Generated***
Plaintext Decrypted with I_1	Did you ever hear the tragedy of Darth Plagueis The Wise? I thought not. It's not a story the Jedi would tell you. It's a Sith legend. Darth Plagueis was a Dark Lord of the Sith, so powerful and so wise he could use the Force to influence the midichlorians to create life… He had such a knowledge of the dark side that he could even keep the ones he cared about from dying. The dark side of the Force is a pathway to many abilities some consider to be unnatural. He became so powerful… the only thing he was afraid of was losing his power, which eventually, of course, he did. Unfortunately, he taught his apprentice everything he knew, then his apprentice killed him in his sleep. Ironic. He could save others from death, but not himself.
Plaintext Decrypted with I_2	Did"you evf< hear tieItrageey!og!>arth Qlagueks The X3se? I tio□ght npt/ Ju–s not!a stoty the K/di woule □ell ypu/ Ju–s a Sjth leiend. Db<th Plagve□s was!a!Dbse Lord!of thg Sith,!=o powergu□ and to!wjt_ he cpuld uue the G9rce to jn□luencf uhf!gidichmorianu to crf+te life· qe had!svci![knowmedge qf the e+rk side!t□at he!cpumeeven leep tje ones!2e cared!a□out fson ezcng. Tie darm side p0 the Fosc□ is a!pbtix[y to nany adilitiet some coos□der tp ce!vhnaturbl. He"became!=o powergu□… the!oolz!nhing ie was"afraid!9f was lps□ng hit qoxfl, whidh eveptually- of courteU he djd/ Vo`ortunbtely,"he tauh2t his aqp□enticf fvfssthing!he kngw, theo his appse□tice limlfehim io his uleep. J<onic. Hf □ould tawe!pnhers grom dgath, bv> not hins□lf.00101
Plaintext Decrypted with I_3	AravgwKf©gl%hc3H#v4ePDz□ `□□¦S¤dTB□o¥iS~¡bjyhkwY□i $[dJ□h"NRpcl§gh¡□puaQw'sH¡ q¥N□"pKmly s□fRO/ikKHGugf _k□¦ ¡^Rv`Yr-□

Image Key Space	*Plaintext Generated*
	□□xm!ma`elj]&ud>x□NOt□hre □t ¨Y£"□QH\□lNz□t□R£cud5 Qks=^ juN□ny□m6p□3□dM£□ ©7¢5!d□)□□£hd"tsdwe9$zb□! K¤- cg□iz□l□□Mv□^Rk□□□g□□ ¤olapD!pDLb¦i^v□Lop□b†H6% □/ePs¢□bP_K□□F¦□□3dmnh wgf#;,s Ipj□cp□]vO4e"bsª7□ e§□oM□□gt&xfe q□i¤'dlP□dz□e`cmxu□¤□ O^© k□□^!□hj"d7Ej km□gQpc!w□`$Ejs_□;¢ □!qb¢i¤]£H¥qP□□mz 33;;opn□I x□□_$□qmt□a9 ¤□Q□□¨p□□ ¨u¥b□. Kb!ccc□o□&krM□@O□p5E□ ω¤b□O□¡□HNn□d□oP□□ x7r 52q□h□ fjN¢bvOk<Bj==R3□¤ ¦¡p□kYN¬n□□□ ewglxrf□q®3v□N5t¥Co_]-m□ □□□\PN□6p ¥Kr□xan{,-fc- ta¢j2uT□iy- `?lr6A□j□□J□§_¡y □□Bj!k8$> qeJ-#□f□qp□£- eqnld□um□aQ□□□]□ □□q#□□Rjgu uo^f□,# osphcLH`□jz□3L□□v□ □s□`¤IPfw?j)aa^ljW!□wn%nm F kk9s□<n])ZZ]c
Plaintext Decrypted with I₄	Cmbult;¥:r$□□.x- t9dIsG□e□□¨!¥fUE□¢¤lOy□bd K9eE!V□h\3yimO"ugo□f;§ ¡¡¦cPE¢–□M□EyLd%s\|p>{ xiiLO2k□Oquuoe"¥gs□¥□□`N x—rQ□(□e§7"Dg`g<i]%pf=}□ QRu□dr`□l$¢/uM`Vua£7OG¢J □I¢8#zk^"p¥o`(rrP□iw□m5F□ i□dOp□ ª9¥^Of□&□□¨<i(ztj u□eLEkseeJ¤-jlbkG□>□□Ow□ cSf□`□k□□¥vhellqn- 9¡:ax□M9oee@qd □_□J¢v□hUbR□□s¦□□ed; k8!v□hi+xoNmoff- t□`GSh□R□¤¥h□□£□D%□hjp (u4g soi□%5lR□[xeg!c□q|¡□□□□N 2}em□_(□d□ f9td"An□jLt1)¨□g)zlo^□m□ cMo□¥h¨- ¨§GS□□@|&d[k8p¥p□{v□□_ □j=D□3m £lOb□ §g□_¨{¦□□ %Pj!ged□m□op"plM□pgsh.. ¤8□J□q wSm□e¤oV□□$veq]c q`?□ ojT¤.y- l@r□m<Qf□□Oq¥w□s^P§l□□

Image Key Space	*Plaintext Generated*
	□!bL9jFvc□ox1 ujN]uwsreU;□ □□□cPQ□_□□¢Ks□wjl□- j0$ue¡l5{R□cy dqr£gu¡e□□I□¤3vus□□v□□4" Cp^y %£m□s q□¤"j¤mo`□mm□3"□h¢□e□ □d □□S:ly#ln1l¡5TQqr□□].Q C4 □>}□dOp□v□ ¡m□c¦yT□¥?r(ifftiY □tp!poq □gnqak8^

The table above allows for comparison between the plaintext after decrypting it with the key space from I_1, I_2, I_3, and I_4 respectively. Since I_1 was the image that was used to encrypt the plaintext, the decrypted ciphertext returned the exact same string as the original plaintext, proving that the encryption/decryption process works. After referencing Table II, the similarities between the key space of I_1 and I_2 are very similar so a lot of characters will be decrypted properly. The decrypted plaintext from I_2 is still mostly unreadable which allows for some security of the original message but having correct portions of the key can make guessing the other portions easier. The decrypted text from I_3, the different image with more segments (key space keys), is completely unreadable. The decrypted text using I_4 is also unreadable because even though the hues are similar, the key spaces are unique enough for this to occur. This proves that unless the same image is used or insignificant filtering is done on the correct key image, the probability of decrypting the plaintext correctly or with enough correctly decrypted characters is low.

REFERENCES

[1] Assignment 3 for Computer Vision

[2] Rastogi, Sachin, “Convert RGB to hex color code in Python” [Online] Available: https://www.codespeedy.com/convert-rgb-to-hex-color-code-in-python/

[3] Dworkin, Morris, “Recommendation for Block Cipher Modes of Operation: Models and Techniques” Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[4] Scikitlearn, “Sklearn.cluster.estimate_bandwidth” Avail.: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.estimate_bandwidth.html

[5] Lucas, George, “Star Wars Episode III: Revenge of the Sith” https://imsdb.com/scripts/Star-Wars-Revenge-of-the-Sith.html

[6] Geeks For Geeks, “Block Cipher Modes of Operation”, Available: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/

[7] https://www.researchgate.net/publication/2997961_An_introduction_to_Block_Cipher_Cryptanalysis

[8] A. Abusukhon and S. AlZu’bi, "New Direction of Cryptography: A Review on Text-to-Image Encryption Algorithms Based on RGB Color Value," 2020 Seventh International Conference on Software Defined Systems (SDS), 2020, pp. 235-239, doi: 10.1109/SDS49854.2020.9143891. https://ieeexplore.ieee.org/abstract/document/9143891