Seeding Encryption Key Generation using Images

Jonathan Garner & David Wynne

Encryption

- Converting data into a code that can only be decrypted by the intended party
- Random number generation is an important component
- Keys to encrypt/decrypt data need to be randomly generated so malicious parties cannot guess the keys
- Computers use pseudo-random number generators (PRNGs)
    - Truly random number generators (TRNGs) rely on a real world source
        - E.g. Radioactive decay, Atmospheric noise, etc.

OpenSSL

- The most commonly used implementation of the SSL/TLS protocol
  - Used for transferring data securely through the internet
- Has four components
  - Symmetric key (secret key) encryption
  - Asymmetric key (public key) encryption
  - Message Digests and digital signatures
  - Certificates
- RSA
  - Asymmetric
  - Pair of keys
- AES
  - Symmetric
  - One key

Seeding

- Random number generators need to be given a initial state
- This initial state is called a seed and must also be random
- OpenSSL uses a 256 bit long seed to seed its PRNG
    - Typically uses random keystrokes from a users keyboard or other similar sources
- Our goal was to instead uses images as an input as this could have more robust applications for mobile devices

Images as a Seed

- Converting the intensity values to bits to make a seed
- This project used JPEG images as input
- The compression algorithm JPEG uses removes a significant portion of the randomness in the original image
- The algorithm was designed to take this weakly random source and turn it into a truly random source

Testing

- Random Number Generators need to go through a series of statistical tests before they can be declared random
- The recommended suite of tests by OpenSSL is NIST's STS
- We used this suite to iteratively design the algorithm to pass each of the tests in the suite
- Tested ~1,000 images (256,000 bits)
    - Passed the 12/15 tests
    - The remaining 3 required a longer input length

The Algorithm

- There were three main components to the algorithm
  - Image Processing
  - Bit Manipulation
  - Hashing

Image Processing

- The largest component of this project was testing the best image processing techniques to use to create a TRNG
- The techniques that were used in the final algorithm were
    1. Bad image detection
    2. Filtering
    3. Matrix manipulations

Bad Image Detection and Adjustment

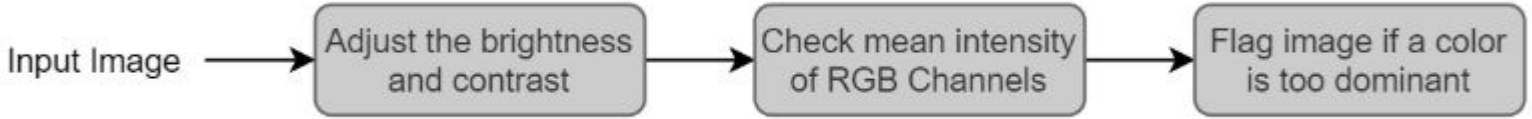


Figure 6. Overview of Bad Image Detection and Correction Algorithm

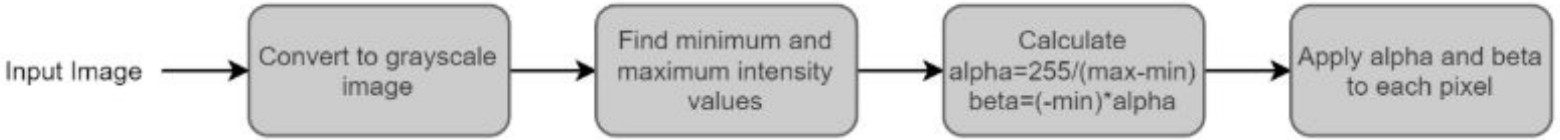


Figure 7. Overview of Image Adjustment

$$outputPixel(row,\ col,\ channel) = alpha * inputPixel(row,\ col,\ channel) + beta \qquad (1)$$

[10]

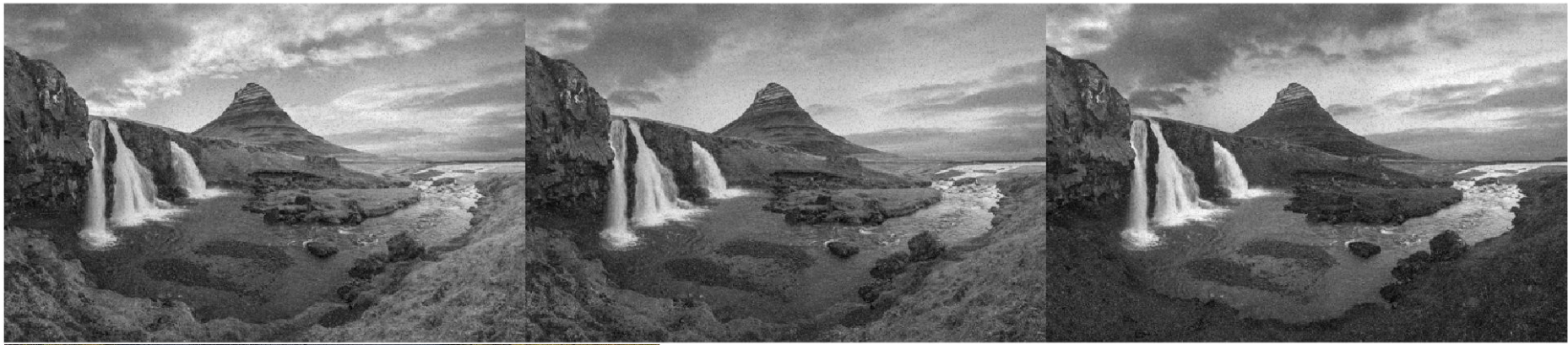
Image Adjustment

Figure 10. Example of modifying a low quality dark image. The left is the original, the middle is after salt and pepper filtering, and the right is after auto adjustment. The salt and pepper filter can be seen best in this example. Alpha = 5.43 and Beta = 0.00

Figure 11. Example of modifying a light image. The left is the original, the middle is after salt and pepper filtering, and the right is after auto adjustment. Alpha = 3.27 and Beta = -572.16

Filtering

- Through testing determined that a Salt & Pepper filter was the best filter to use
- Applied the filter to each of the RGB color channels individually

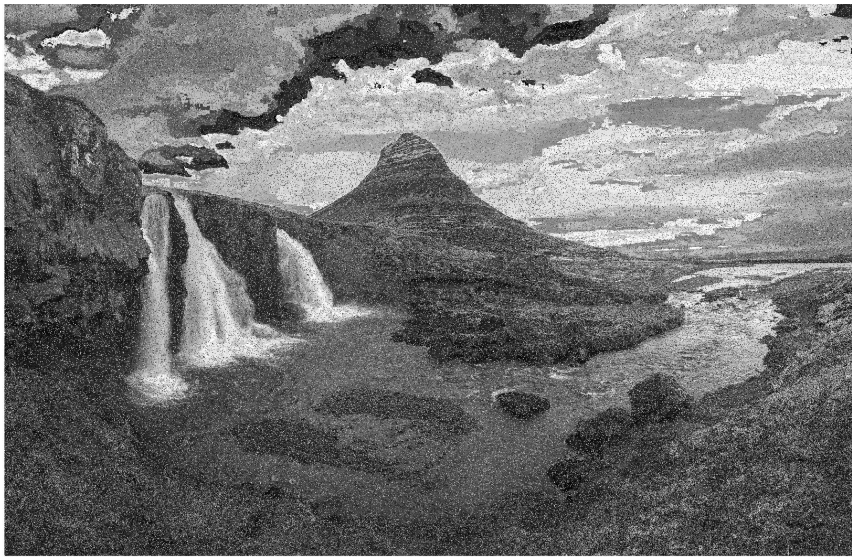


Matrix Manipulations

- RGB color channel blending
- Converting matrix to a square
- Converting matrix to one-dimensional array

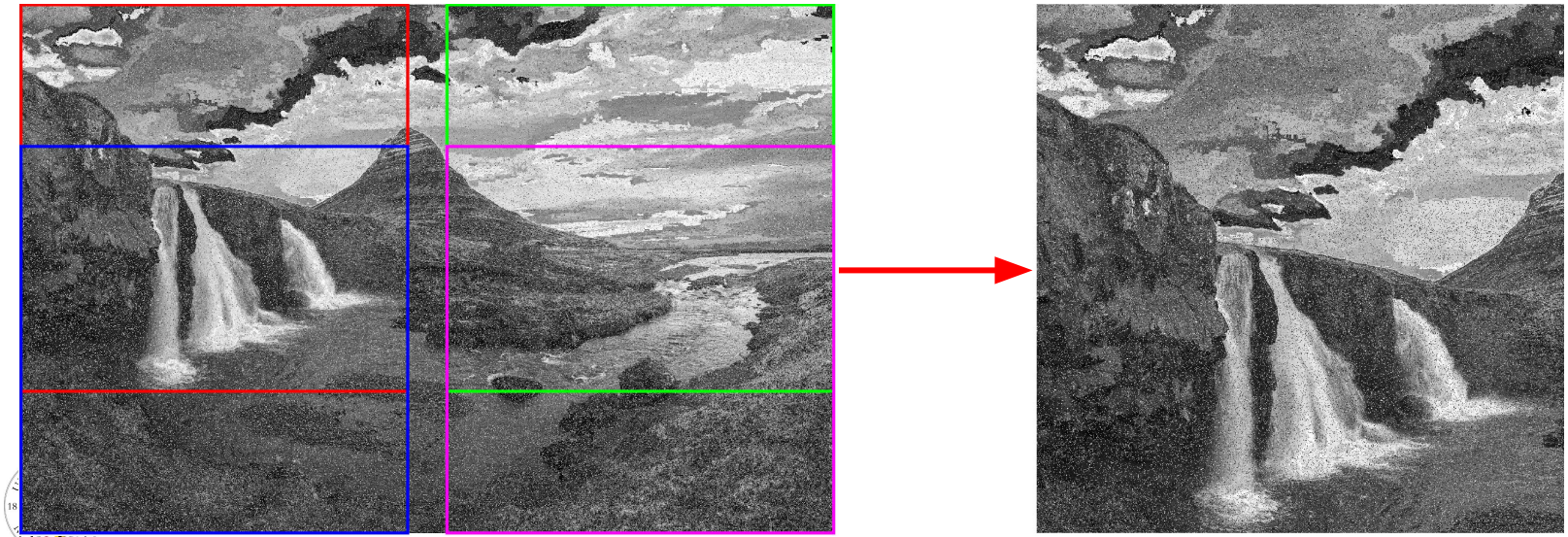
RGB Color Channel Blending

- Took each pixel for the three channels and XOR’d them together
- Combined with the filtering above this takes a colorful images and makes it very noisy



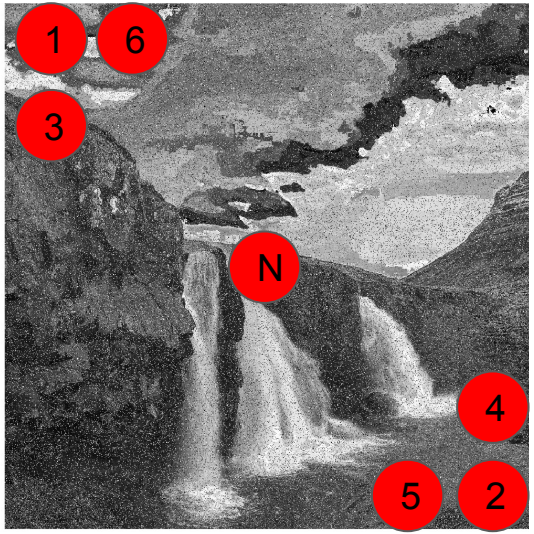
Converting Matrix to a Square

- Was done to make the logic easier and further alter the image
- Cropping and reshaping were tested
- Cropping was better



Converting matrix to one-dimensional array

- Also done to simplify logic and alter the image
- Multiple methods were tested
- The final chosen method was “jumping” through the matrix
- If the matrix is NxN this will generate an N^2 1D array

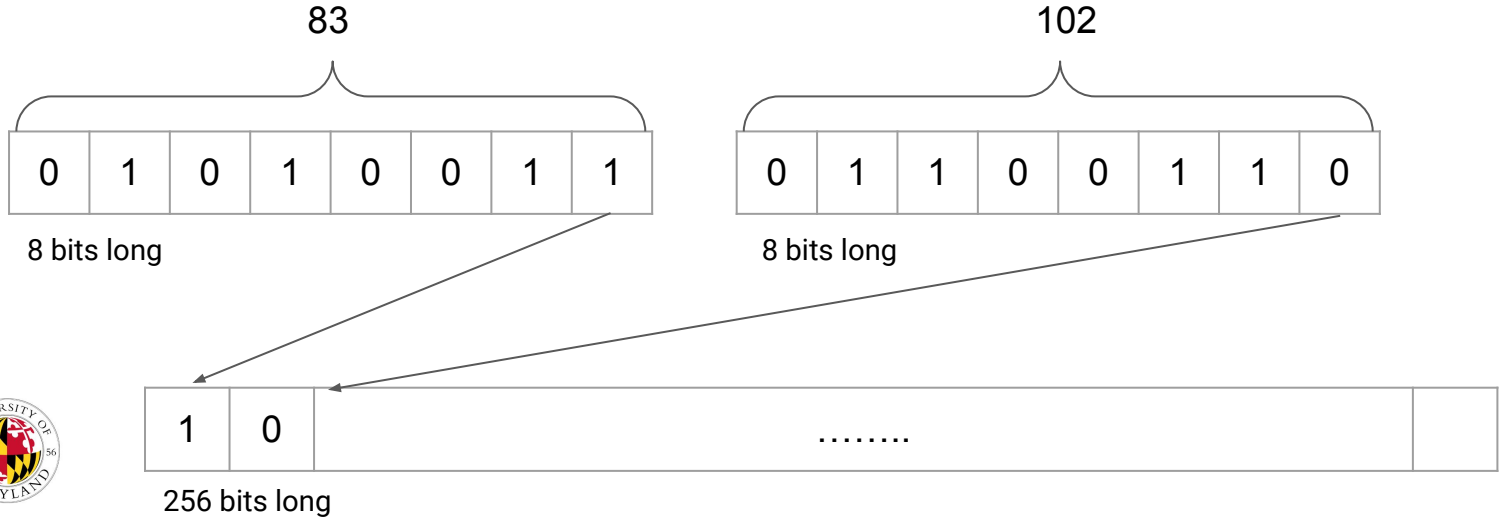


Bit Manipulation

- Generate bit strings
- Extractors

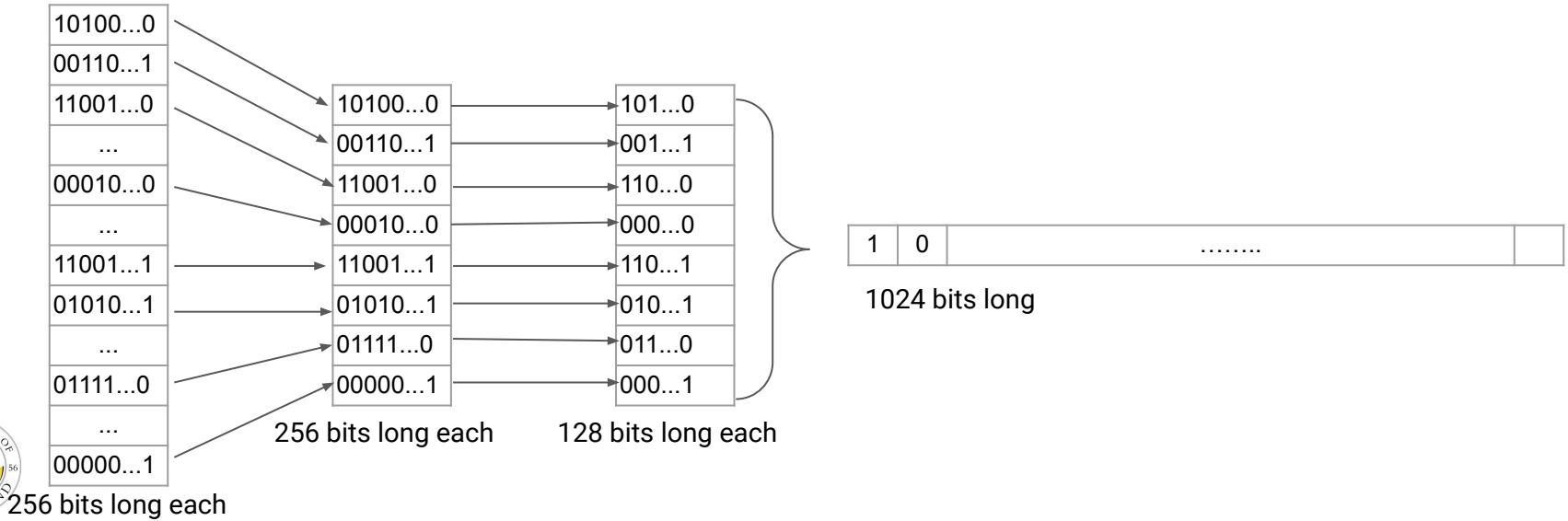
Generating Bit Strings

- Tested how many bits of each pixel to use
- Chose to use LSB
- Provided consistent spread of 0s & 1s
- For an N^2 1D array, this process generates $N^2/256$ bit strings of 256 length



Extractors

- Used to aid in extracting randomness of a weakly random source
- We XOR'd two successive bits to get the set of bits that would be used in the next step
- Eight strings were randomly chosen from the collection of strings to generate a 1024 long string that is used for hashing



Hashing

- The final push to make a TRNG
- Slightly different inputs result in very different outputs
  - This aids in randomness
- Used SHA-256
- Takes a 1024 long input and produces a 256 long output
- The output is the seed that is passed onto OpenSSL

Seeding OpenSSL's PRNG

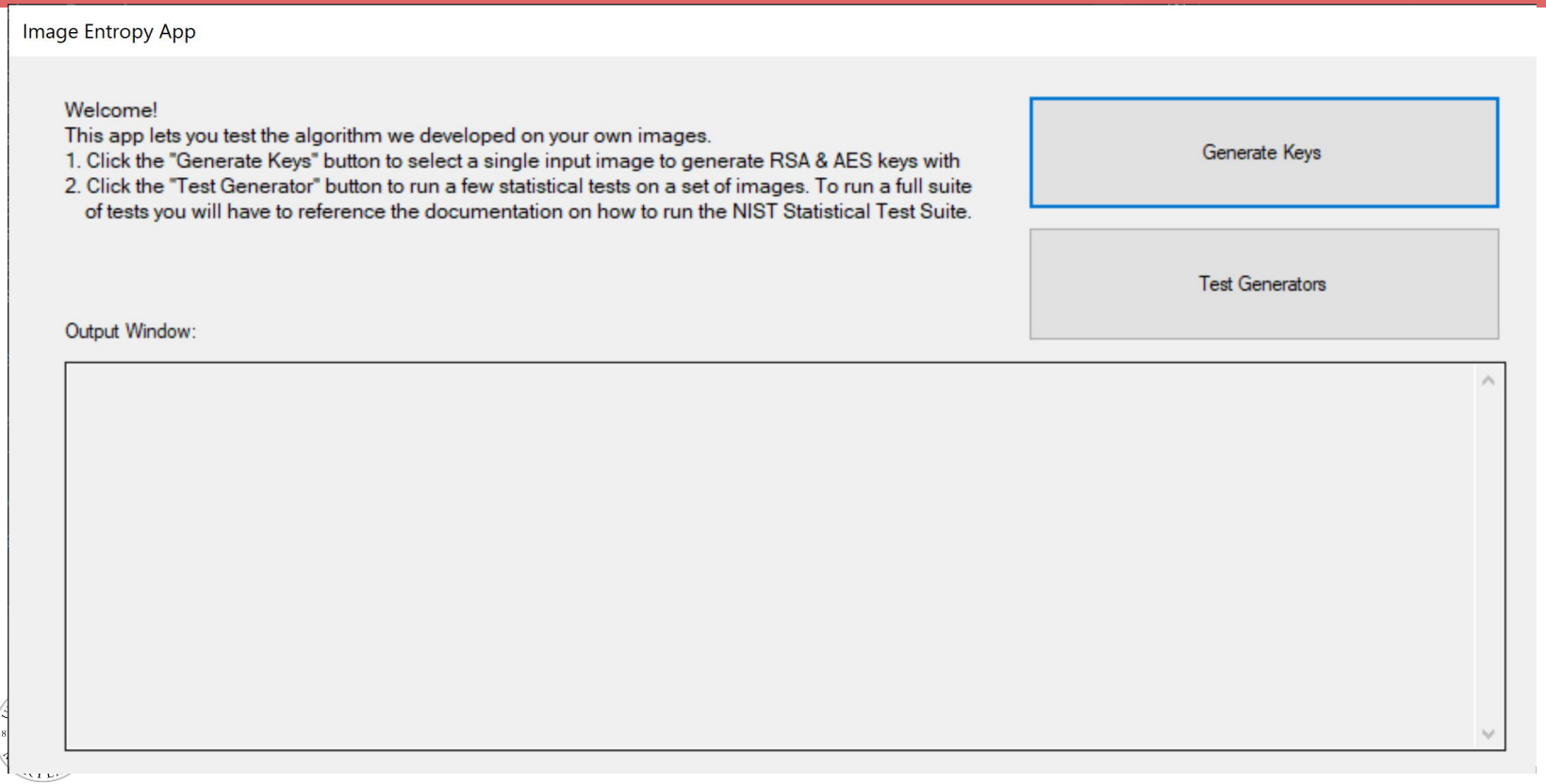
Seed:
00111111000100101111100000000000
11011010101000100100101011000011
11110010001010110100101110001100
11000010110101101011100101000010
01100001101110001101010011000000
10101010000110000110100010100000
11001111000010101000100110110101
00111000011010001001001110110101

OpenSSL PRNG

RSA Key Pair

AES Key

Prototype Application



Prototype Application Demo

Applications and Expansions

- Applications
  - File transfer app
  - Anywhere an image can be inputted by a camera
- Expansions
  - Video
  - Accelerometer or Gyroscope
  - RAW Image data

Summary

- The TRNG developed in this project is a good foundation for further work
- Images are a flexible source of random data
- Project inspired by Cloudflare's LavaRand
- The reliance on image processing techniques allows for an easier to understand and more modular encryption algorithm

References

[1] https://wiki.openssl.org/index.php/Random_Numbers#Generation
[2] https://en.wikipedia.org/wiki/Lavarand
[3] https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/
[4] https://www.visolve.com/ssl.html
[5] https://simple.wikipedia.org/wiki/RSA_algorithm
[6] https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm
[7] https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard
[8] https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm
[9] https://en.wikipedia.org/wiki/JPEG
[10] https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html
[11] https://en.wikipedia.org/wiki/Randomness_extractor
[12] https://link.springer.com/chapter/10.1007%2F3-540-39799-X_41#page-1
[13] https://en.wikipedia.org/wiki/SHA-2
[14] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

References

[15] https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software
[16] https://www.cygwin.com/
[17] https://en.wikipedia.org/wiki/Photoelectric_effect
[18] https://en.wikipedia.org/wiki/Raw_image_format
[19] http://www.cs.tufts.edu/comp/116/archive/fall2013/pnixon.pdf
[20] https://link.springer.com/chapter/10.1007/978-3-642-25541-0_58
[21] https://wakespace.lib.wfu.edu/bitstream/handle/10339/62642/Li_wfu_0248M_10943.pdf

Summary

- The TRNG developed in this project is a good foundation for further work
- Images are a flexible source of random data
- Project inspired by Cloudflare's LavaRand
- The reliance on image processing techniques allows for an easier to understand and more modular encryption algorithm

Backup

SSL/TLS Protocol

- **Record Protocol:** Used for encapsulation of various higher-level protocols.
- **Handshake Protocol:**
  - Allows the server and client to authenticate each other
  - Negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.
- Asymmetric key encryption is used in the handshake
- Symmetric key encryption is used for the actual data transfer

Asymmetric Encryption

- Uses a public and private key
  - Public key is accessible by anyone
  - Private key is only known to the owner of the key
- Need both the sender’s public key and the receiver's private key to decrypt the data
- Most widely used is RSA encryption which uses large prime numbers to generate the public and private key

RSA

- Asymmetric Encryption Algorithm
- Uses large prime numbers to generate public and private keys
- Data transfer requires both public and private keys for decryption [5]

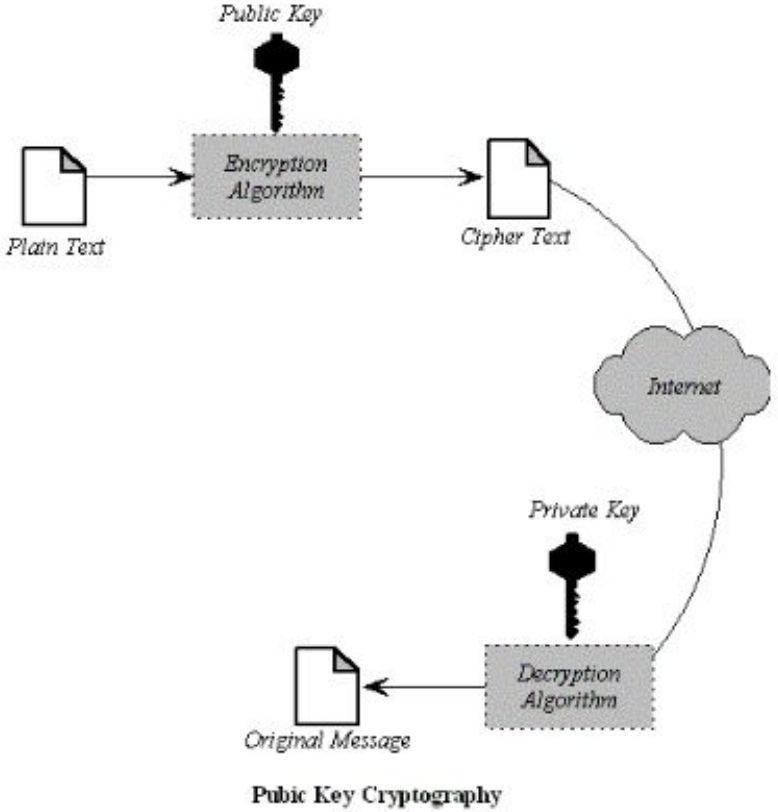


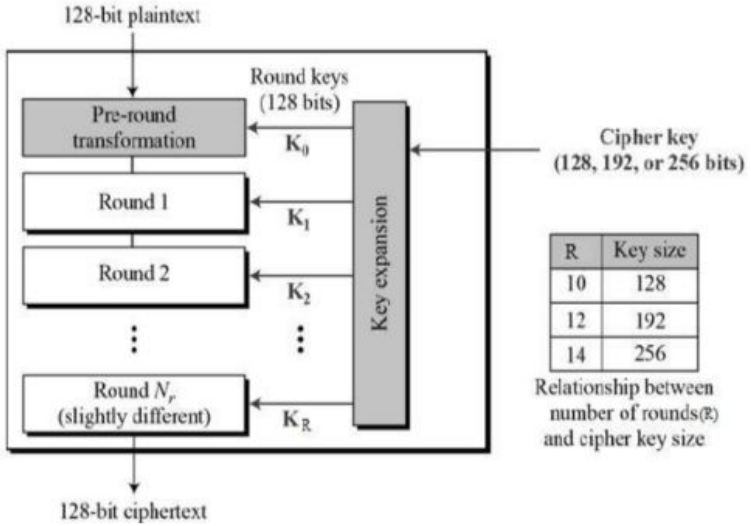
Figure 2. Standard flowchart for asymmetric encryption. [4]

Symmetric Encryption

- A single key that is used to both encrypt and decrypt data
- This key needs to be securely transferred since access to it prevents secure encryption
- In SSL/TLS asymmetric encryption is used to transfer the symmetric key
- AES is the most commonly used symmetric encryption algorithm

Advanced Encryption Standard (AES)

- 3 Block Ciphers
  - AES-128
  - AES-192
  - AES-256
- Uses keys of varying bit length to encrypt and decrypt blocks
- Each cipher completes multiple rounds of transformations [7]



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Figure 4. Schematic of AES Structure for a single cipher block. N rounds are performed in the cipher. The cipher key length corresponds to the current cipher block. In the table on the right, R corresponds to the number of rounds for a certain key size. [8]

JPEG Compression

- Takes many shortcuts based on what the human eye can detect as a change
  - Downsamples Hue & Saturation
  - Reduces frequency information in high frequency components during Quantization

Filter Testing in MATLAB

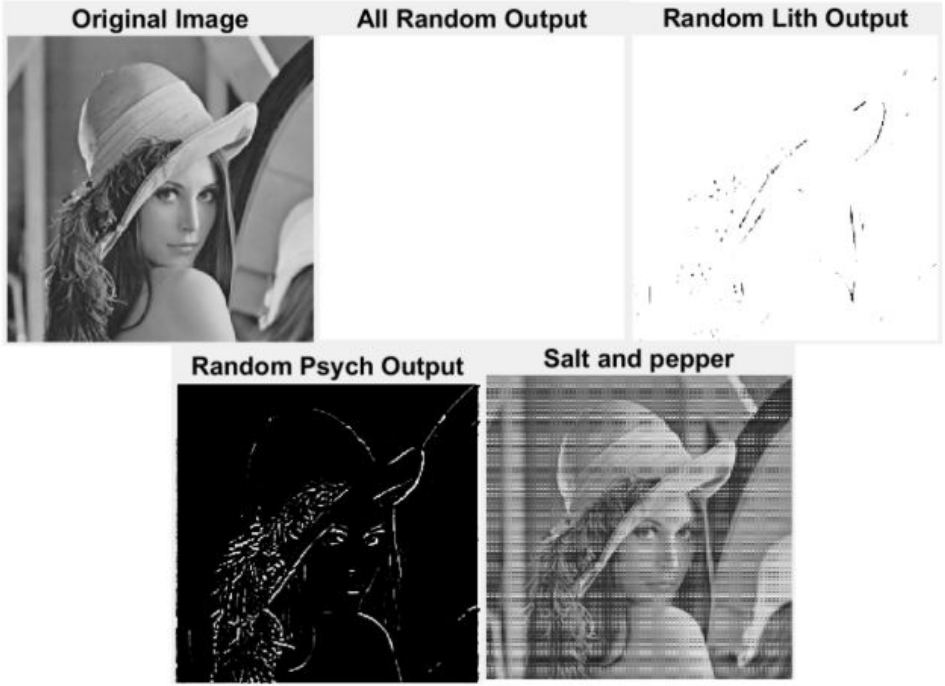


Figure 14. Results of four different filtering techniques. Besides the salt and pepper filter, each filter was 5x5. The random values for the filter or the salt and pepper filter were [1, 10].

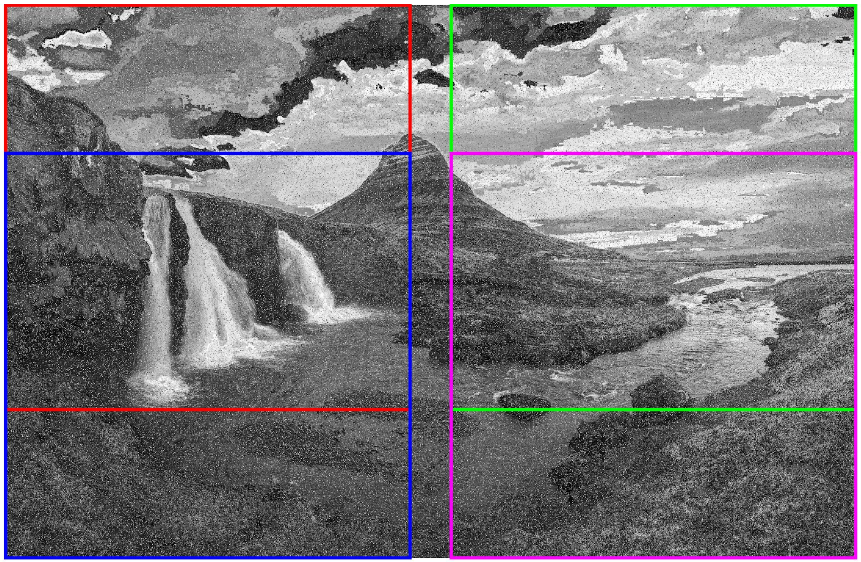
```
-1 -1 -1 -1 -1
-1  A  B  C -1
-1  D  E  F -1
-1  G  H  I -1
-1 -1 -1 -1 -1
```

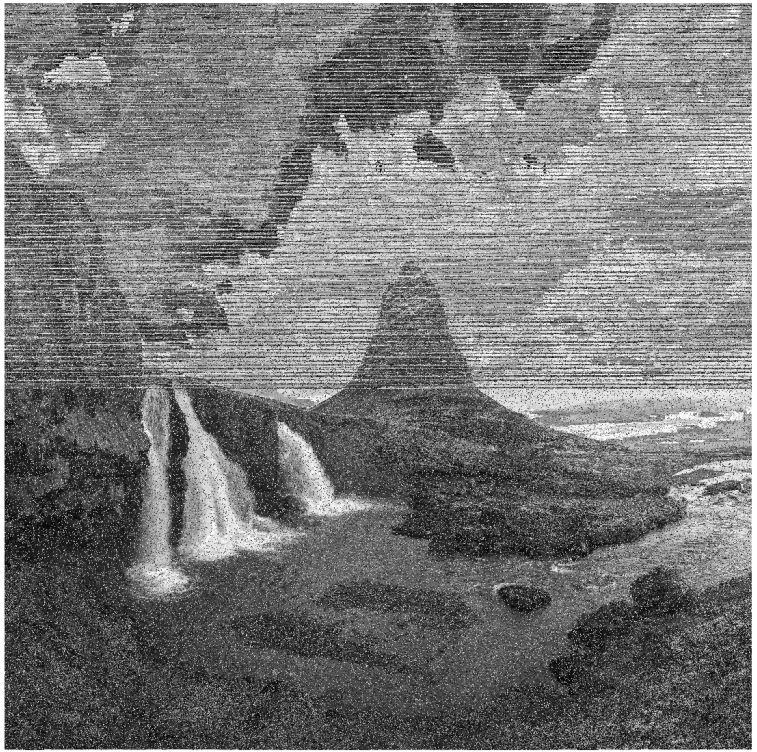
Figure 12. Template for Random 5x5 Lithographic Filter

```
0 -1 -2 -3 -4
0 -1  A  B  1
0 -1  C  D  1
0 -1  E  F  1
0 -1 -2 -3 -4
```

Figure 13. Template for Random 5x5 Psychedelic Filter

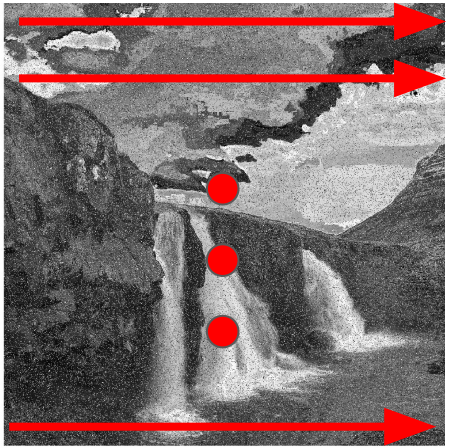
Cropping vs. Reshaping

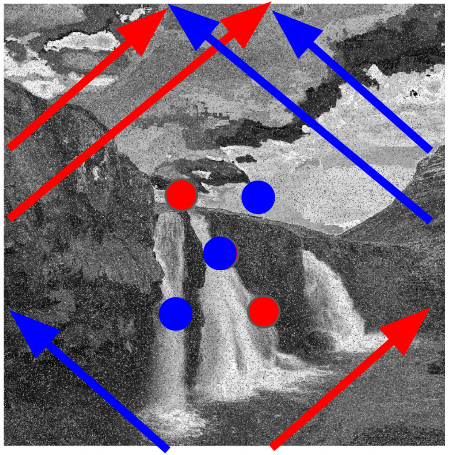


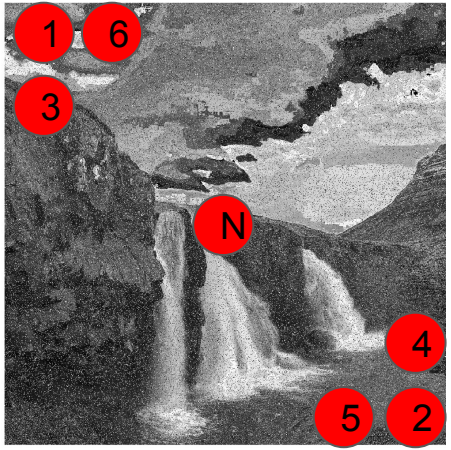


Matrix Division methods

- **Snaking:** Horizontally weaves through the 2D matrix row by row
- **Crossing:** Follows the diagonals of the 2D matrix in a criss-cross fashion
- **Jumping:** Jumps back and forth between the top left and bottom right corner until it has reached the center







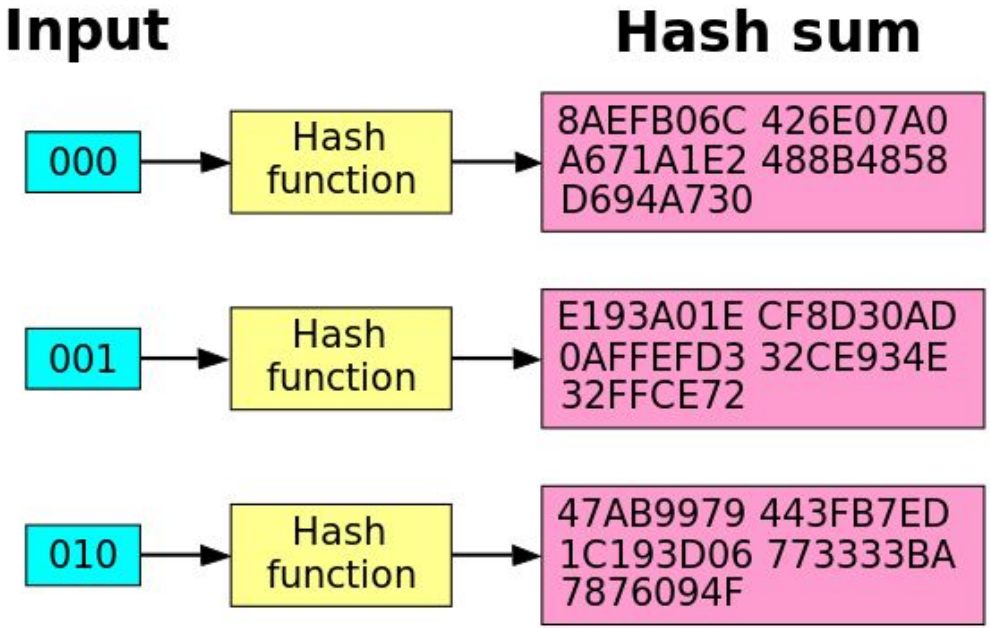
Bit String Generation Methods

- All 8 bits of the 0-255 intensity value of the pixel
  - Favored zero too much especially when being put into an extractor
- High-low threshold
  - Greater than 128 would be a “1”
  - Less than or equal to 128 would be “0”
  - For darker images this also favored zero too much
- Taking one bit of each pixel
  - For the first pixel it would take the LSB, second bit one to the left of the LSB, and so on
  - This still generally favored zero
- Taking the LSB
  - Proved to have the most equal distribution of 1s & 0s

Von Neumann Extractor

- Takes two successive bits
- If and only if the two bits are different it outputs the first of the two bits
  - 10 = 1
  - 01 = 0
  - 00 = no output go to the next two bits
  - 11 = no output go to the next two bits
- This helped but ended up working against the LSB generation which was also working to normalize the distribution of 1s & 0s

The Avalanche Effect in Hashing Algorithms



Example of Avalanche Effect for SHA-1
https://en.wikipedia.org/wiki/Avalanche_effect

Random Selection

- Want to select from N options or bit strings multiple times in the project
- Calculate the number of bits, B, to represent a number N
- Measure the execution time of a function
- Use B bits from the execution time to produce a number from 0 to N

Testing

Summary		
	No Hashing	Hashing
Approximate Entropy	FAILURE	SUCCESS
Block Frequency	FAILURE	SUCCESS
Cumulative Sums	SUCCESS	SUCCESS
FFT	FAILURE	SUCCESS
Frequency	SUCCESS	SUCCESS
Linear Complexity	SUCCESS	SUCCESS
Longest Run of Ones	SUCCESS	SUCCESS
Non-Overlapping Template	MOSTLY SUCCESS	SUCCESS
Overlapping Template	FAILURE	SUCCESS
Rank	FAILURE	SUCCESS
Runs	FAILURE	SUCCESS
Serial	FAILURE	SUCCESS

Gantt Chart

Project Start:

Display Week: 1

TASK	ASSIGNED TO	START	END
Research and Planning			
Research OpenCV	Both	3/23/20	3/24/20
Research encryption keys	Both	3/23/20	3/24/20
Download, Install OpenCV, prototype	Both	3/24/20	3/28/20
Manipulate long bit strings	Both	3/24/20	3/28/20
Get pixel values from image	Both	3/24/20	3/26/20
Image Processing and Bit Manipulation			
Matrix division	David	3/29/20	4/5/20
Filter testing and implementation	David	3/31/20	4/5/20
Bad image correction	Jonathan	4/6/20	4/28/20
Bad image detection	Jonathan	4/10/20	4/17/20
Bit Extractor	David	4/13/20	4/16/20
Make image square	David	4/15/20	4/29/20
Encryption, Testing, Hashing			
Testing Randomness	David	4/12/20	4/29/20
Add Hashing Algorithm	David	4/15/20	4/17/20
Generate RSA and AES keys	Jonathan	4/19/20	5/2/20
Windows App Development	David	4/20/20	5/2/20
Seed keys with our data	Jonathan	5/2/20	5/3/20
Report and Presentation			
Report	Both	4/25/20	5/13/20
Presentation	Both	5/3/20	5/13/20
Code cleanup	Both	5/13/20	5/14/20