***Science Scholars Daily Log 2016***

*Federico Reyes Gómez*

Friday, June 24, 2016

Checked if the code I already had for the actual automata process was correct. Started by writing a function that could reverse the forward\_rule\_list so I can reverse the Cellular Automata. One of the main reasons I decided to use this block cellular automata was the fact that it was easily reversible and I didn’t have to make it second order to do that. Started to figure out how to reverse the forward evolution. Instead of running margolus1(forward\_rule\_list) and margolus2(forward\_rule\_list), I have to run margolus2(reverse\_rule\_list) first and then margolus1(reverse\_rule\_list) in that order. This reverses the evolution to (hopefully) get back to the original data. Finally, started thinking about how to go from the evolved dish back into a file. I realized I need to (1) turn dish into a string bit sequence, (2) for each 8 bits, turn into byte number, (3) turn the byte number back into unicode using chr(), and (4) write to file.

Sunday, June 26, 2016

Succeeded in writing the write\_dish\_to\_file function. I ran the script and I succeeded in evolving the file using the cellular automata and then successfully reverting it back to the original information. Now I have to see if this will scale to bigger files; i will probably have to figure out how to do it by blocks at a time. I realized that the time required to open the file (and probably to encrypt) varies directly with the file size.

Monday, June 27, 2016

Packaged my script into a python module and I uploaded it to PyPI so I can now “pip install block\_ca”. Next I would need to update the documentation to make it easier.

Tuesday, June 28, 2016

Added a progress bar to see % completion when the evolution takes a while. Also started working on including multiprocessing in the program to speed up the evolution.

Wednesday, June 29, 2016

Decided I should split the input into blocks to process them separately. Wrote a script to graph the time taken to iterate through a block of different sizes.

Thursday, June 30, 2016

Ran file\_size simulations to determine best block sizes

Friday, July 1, 2016

Finished the file\_size simulations. Realized that time varies proportionally to number of generations. Time taken to open file also varies proportionally to the file size. Chose 256 bit blocks which aren’t huge but aren’t too small.

Wednesday, July 6, 2016

Started modifying my code to accommodate blocks. The main function i needed to write is the blocks\_to\_dish function and an open\_file\_blocks function to make blocks from the start.

Thursday, July 7, 2016 to Tuesday, July 12, 2016

Took a break from Science Scholars to volunteer in the city for Safe Passage Project, for which I sometimes also volunteer during the school year. They help provide pro-bono legal aid to undocumented minors and my fluency in spanish puts me in the perfect position to go and translate, make calls, accompany clients, etc.

Wednesday July 13, 2016

Fixed an issue with the block\_ca module where I couldn’t upload to PYPI and i couldn’t merge with the local branch. I wrote a script to automate it but now I can’t edit anything on github directly. I have to do all my edits on cloud9.

Thursday July 14, 2016

Wrote a test script to calculate the avalanche percentage, hamming distances, and entropy. Realized my algorithm wasn’t working and started reading [a thesis](http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd) on the use of margolus cellular automata in AES encryption

Saturday July 16, 2016

Finished with the block\_ca updating so now I need to read the thesis further. Wrote down some notes below ( [1] )

Sunday, July 17, 2016

I worked on creating the key using key expansion. I read some of my past outlines and the thesis.

Monday, July 18, 2016

Succeeded in creating a nonce. First, a completely random 256 bit block is created using os.urandom(32). Next, the user supplies a 32 character password which is converted to a 256 bit block. These two blocks are XOR’ed. Next, the key (which consists of the rule\_list and the number of generations) is encoded into a 256 bit block. If the key isn’t that large, the block is padded with random bits(with the pad length at the end to undo the padding). The key block is than evolved with the life-like “fredkin” rule (B1357/S02468). Finally, that evolved key block and the previously XOR’ed password and rand blocks are XOR’ed together. 

Tuesday, July 19, 2016

Wrote in the nonce into my code. The first block (“dish0”) is XOR’ed with the IV before evolution. The second block is XOR’ed with the previously ciphered block (encrypted “dish0”). The third block is XOR’ed with the previously ciphered block (encrypted “dish1”) and so on. 

Then to reverse this, you have to start with the last block (say “dish5”), **evolve it first**, and then XOR it with the next block (“dish4”). This is repeated until you get to the last block to be processed (“dish0”) and XOR it with the IV after evolving it backwards.

Wednesday, July 20, 2016

Weeded out some bugs in the code I finished yesterday and started to run simulations and tests. Some results are promising and others aren’t. Entropy is almost always ideal at 7.99\* and hamming distance of the encrypted file is .50 +/- .03. Avalanche percentage, however, which I need to be at 50%, varies wildly going from 0.5% to 13%. I need to figure out how to make some part of the algorithm more sensitive to initial conditions

Thursday, July 21, 2016

While watching TV, someone in the show mentioned a checksum error so I looked it up and it turns out that it might just be exactly what I need. A checksum is a hashing function so I looked up what are the good hashing functions with existing implementations in python and I found MD5. MD5, however, is vulnerable to brute force attacks so I ended up using SHA256 since I need a 256 bit block exactly. I’m going to XOR it with the final IV and see if that increases the avalanche percentage.

Friday, July 22, 2016

Found out that the average difference in the hashes with one bit flip is 50%, but that doesn’t seem to translate over to my general algorithm for some reason. It’s probably due to the fact that I don’t change the key when evolving the flipped file. That’s why the avalanche percentage varies so much. When the flipped bit is towards the end, the change is minimal because the first blocks aren’t affected by it. When the flip is in the beginning, the % is much higher because all the blocks are then affected by the change, proving the utility of using the previously ciphered block to cipher the next. This decreases the parallelization of the algorithm (you need to evolve the blocks in a certain order), but the security purpose makes up for that.

Sunday, July 24, 2016

Finally was able to keep the rand\_block constant to see if the design of the key was enough to satisfy the avalanche requirement and voila, it worked! I’m getting entropy of 7.99 ± .02, hamming distances of 0.500 ± .008, and finally avalanche percentages of around 50% ± 03%! Maybe now I can consolidate the code into one file, comment it all, and then focus on 1) adding multiprocessing to the block\_ca and base margolus evolution to speed it up and 2) make the code sleeker and more efficient to increase performance.

Monday, July 25, 2016

Today I wrote a function to calculate the flatness of the byte histogram by calculating the average distance of the actual frequency from the ideal frequency (1/256). I tried to use multiprocessing but it didn’t work. Read some of the articles again to verify what further tests I have to do. My main focus for tomorrow will be to find a way to either code or use pre-made code to analyze my ciphertext using the ENT, NIST, and DIEHARD tests. [Good website for those](https://gerhardt.ch/random.php)

Tuesday, July 26, 2016

I spent all day looking for implementations of the test suites that I could use to no avail; there’s nothing out there that’s easy to use. I’ll look for one more day but I think I’ll have to find someone that understands another language to help me translate from a java or C implementation

Wednesday, July 27, 2016

I’ve been trying to send an HD video file to my friend in New York but the internet connection here is terrible. I got the idea to “stream” a file directly to someone else’s computer by sshing into their computer and then writing a new file line by line. This attempts to cirvumvent the usual approach which is to upload to youtube or dropbox or something and than have the other person download it. I later realized you can just use “$ scp foobar.txt [your\_username@remotehost.edu](mailto:your_username@remotehost.edu):/some/remote/directory”

Thursday, July 28, 2016

Spent all day working on the applescript app to send a file remotely. The script takes the user’s private password and a shared keyword and outputs a list of bytes. The person sending the file uses that byte list and the shared keyword to have the script generate the password behind the scenes and use it to log in to the other computer using ssh. It than does the scp command. Got it to work but have to test it on another computer except my own.

Saturday, July 29, 2016

Spent the day looking for implementations of the 3 test suites and i realized how clueless I am. [NIST](http://csrc.nist.gov/groups/ST/toolkit/rng/batteries_stats_test.html) released the code for their test suite on their website along with the proper [documentation](http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf) on the usage of the code. The [ENT](http://www.fourmilab.ch/random/) test suite is also available online, I only now need to find an implementation of the [DIEHARD](http://stat.fsu.edu/pub/diehard/) test suite that I can use.

Tuesday, August 9, 2016

Today I implemented a kivy app for block cellular automata which I will attempt to port to ios

Thursday, August 11, 2016

I succeeded in making the margolusCA app and it’s now on my phone. I also started working on a kivy app for the computer just for the encryption algorithm and it seems to be working

Friday, August 12, 2016

Succeeded in making the CA encrypt app to work

Monday, August 15, 2016

Succeeded in making the ent and diehard tests work with brew install ent and brew install dieharder but the NIST test suite is giving me more problems. It asks for a length in the beginning which i dont know what to input and then a number of bitstreams to use. By varying these values it sometimes works and sometimes doesnt so i think i need my mentor’s help

Monday, August 16, 2016

Tried using the NIST python implementation described here: <https://gerhardt.ch/random.php> . It is much easier to use but also much, much slower so I’ll have to see which tests are performing slowly

Tuesday August 17, 2016

Wrote the script using ent, diehard and nist so now I have to include the avalanche tests

Wednesday August 18 2016

Included the avalanche tests and started to run some of the tests

Thursday August 19 2016

Using my new kivy skills, I wrote a GUI app for the Science Scholars Mentor database. Now i need to figure out a way to keep the json online so it doesn’t have to be saved locally with the app and its always backed up just in case…

Sunday August 21 2016

Finished the script to test my algorithm and got it to work on instance 2 on google compute cloud, created a snapshot to save the state. Also got the scholars database app to work by uploading and downloading from dropbox with a new account. All i have to do now is package the kivy file to an exe for use on a windows computer…

Monday August 22 2016

Ran tests today, thankfully found out that if i use tmux, i can keep the scripts running even while i’m not on the ssh and if i use 2 sessions on a 2 cpu machine, both cpus are used. tmux attach -t <session name> allows me to switch

Tuesday August 23 2016

Kept running tests. I ran 3 ‘huge’ files overnight but they still haven't finished

Wednesday August 24 2016

All 3 of those tests failed due to a memory error so I’m just gonna include that as a limitation/try to fix later on, my coding skills and the language used prevented me from implementing the most efficient implementation of the algorithm. I stopped the instances and started 2 8-core ones so now i’m running 8 tests at a time

Thursday August 25 2016

Tests that failed nist:

1. IMG\_5583.mov (no process)
2. IMG\_5580.mov (no process)
3. 88404.pdf (no process)
4. IMG 5544
5. Lotr.pdf

Monday, August 29 2016

Kept running tests

Tuesday August 30 2016

Again, kept running the tests

Wednesday August 31 2016

Wrote a script to manually run the nist suite on files where the pca\_tests didnt work with nist

Thursday September 1 2016

Finished pca\_tests and ran the nist.py on files where nist didnt initially work

Friday September 2 2016

Wrote a script to put the .json information into an excel sheet and spent the day running it with all the files

Monday September 5 2016

I spent all day meticulously organizing my main data into correct columns for analysis

Tuesday September 6 2016

Performed 3 more tests, sensitivity. While creating a new random key every time but keeping it constant for the change: flipped one bit in the key and measured % change, switched 2 numbers in the rule list and measured % change, added/subtracted 1 to the generations and measured % change, flipped one character in password and measured % change (keeping the random block the same this time)

Wednesday September 7 2016 (Day 45)

Finished up the password test and made a chart that I can put in my paper. Finally done with data collection. Next step is writing the paper

Testing:

* On files (1,000 files)(200 images)
  + Frequency Histogram flatness - Should be uniform
  + Avalanche Effect - 47%-53%
  + Final entropy
  + Final hamming distance
  + NIST, ENT, DIEHARD results
* On algorithm
  + Sensitivity to key variation (10,000 generated files)(1000 files)
    - Sensitivity to change in one number?
      * One generation, flip in rule\_list, password character
    - Sensitivity to change in one bit in the fredkin-evolved block?
  + Sensitivity to nonce variation
    - Flip one bit in the fully processed nonce
  + Change in key, one generation, flip rule list

[[1]](http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd) - http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd

* Injective - “The injective property simply requires that all inputs to the function have 8 specific outputs” [[1]](http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd)
* “A CA with a bijective function ƒ can be described as a reversible cellular automata or RCA. “ [[1]](http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd)
* “However, running a RCA on a block of data for an arbitrary number of generations does not make a cryptographically secure algorithm. “ [[1]](http://ir.library.louisville.edu/cgi/viewcontent.cgi?article=1862&context=etd) (35)

<https://www.youtube.com/watch?v=AMH4XrPwmIY>

<https://www.youtube.com/watch?v=nWDqecLnAgE>

<https://www.youtube.com/watch?v=ke80kLGFXds>

<https://www.youtube.com/watch?v=jaTNXFxubyo>