Cryptography Report

January 2021

1 Brute Force Attack with CUDA

My first instinct when approaching this problem was to mount a brute force attack. From doing research online I learned of people's success in using the parallel nature of GPGPU computing [1]. The only experience I had in coding for GPUs before was writing OpenGL code for last year's Compute Graphics sub-module. I spent a week teach myself the basics of CUDA programming, I learnt the nature of the Nvidia Pascal GPU architecture and how to use the GPU API to link CUDA device code and C++ host code efficiently [2].

My implementation of the attack calls 2048 blocks with 1024 threads per block. Although my fairly outdated GPU only has 768 CUDA cores (modern GPUs have upwards of 4,000), CUDA is still able to run in this fashion via efficient context-switching and shared memory usage. In CUDA, threads may be indexed in terms of the block they're in as well their position within the block. This gives each thread (in 1D memory) a thread index $= blockIdx.x * blockDim.x + threadIdx.x$ (i.e. the thread's block number times the number of threads per block plus the thread's position within its block), a number ranging from 0 to 2,097,151. Using this thread index to determine the first 21 bits of a key, each thread need only perform 2^{35} encryptions to cover all 56-bit keys. Even with the decreased clock speed of a GPU this is much more efficient than the 2^{53} keys each thread of my 4 core (8 parallel threads) CPU would need to run.

My initial plan was for each block to have a 14-bit block key (which would require 16,384 blocks), meaning the first 2 parity bits could be computed by the first threads called when the block is instantiated (threads and blocks are called asynchronously so not all threads start at the same time) then stored in the blocks shared memory so later threads need not bother. Although this is possible using CUDA's features described above exceeding the bounds of your hardware too much results in a significant decrease in speed. However, from studying the process of CUDA encryption closer, I noticed that the parity bits are never actually used in any permutation of the key. Thus I was able to modify the permutations so that each key is treated as a pure 56 bit number and parity bits could be ignored entirely.

To perform an attack, a single plaintext-ciphertext pair is needed (which I retrieved using the *encrypt.exe* file). The plaintext and ciphertext are permuted through the initial permutation and final permutation respectively by the host (CPU), this is so that the encryptions later in the attack can take the permuted plaintext as input and compare their output to the permuted ciphertext without worrying about these permutations (speeding up execution). Each thread sets the first 21 bits of its key then begins its encryptions using my CUDA implementation of DES (based on the first technique given in [1]). Once a thread finds the correct key, it announces it to the other threads using device-host global memory.

On my GPU, this attack is capable of roughly 820 million encryptions per second, meaning it would crack DES in just over a year. This number may not be completely accurate as I found it quite difficult to measure the speed of a parallel algorithm. It's even more difficult to estimate efficiency on hardware I don't have access to, but I feel with a decent budget one could build a machine using more modern GPUs that could run this program is a reasonable amount of time.

2 Plaintext Attack

Since the above attack was clearly not going to work with my hardware, I decided to go for a more intelligent approach: Rather than attack the key I decided to attack the plaintext. In a more general problem this would be less effective (there are 2^{64} possible plaintext blocks but only 2^{56} possible keys). However, quite a lot can be deduced about the plaintext in this case. Most obviously, we know the plaintext will consist of three English words separated by "." characters; as no mention of padding was made in the problem specification and the provided encryption oracle only takes inputs with length multiples of eight, I felt confident in deducing that the plaintext is 16 characters long.

I tried obtaining the what3words word list by searching online and contacting their help team directly, however this is proprietary information and they don't give it out. Although I was able to find a decent amount out about it: First, they use 40,000 common English words (not a lot compared to the upwards of 100,000 that

exist); secondly, they only use words of lengths between 4 and 20 characters. This second piece of information is incredibly useful as (not including the "." characters) the plaintext must consist of 14 English characters meaning the words must be length 4, 5, or 6. I downloaded a dictionary of the 70,000 most common English words (in the hopes this would contain the what3words wordlist) and after removing any words of length < 4 or > 6, I was left with only 13,459 words. My fist instinct was to simply query the oracle with all combinations of 3 of these words which would require $13459^3 \approx 2.5 \times 10^{12}$ encryptions in the worst case (already much better than the 7×10^{16} for a brute force attack).

My next thought was to attack each block separately which is more complicated as the middle word begins in the first block and ends in the second. In the case that the first word is 6 letters the other two must both be 4 and the first letter of the second word is last last letter of the first block, there are 6,936 6 letter words in my dictionary and all 26 letters exist as the first letter of a 4 letter word. In the case that the first word is 5 letters the first two letters of the second word are in the first block (and the second word may be 4 or 5 letters), there are 4,265 5 letter words and 248 two letter prefixes to 4 or 5 letter words. Finally, in the case that the first word is 4 letters the first 3 letters of the second word are in the first block and nothing can be said about its length, there are 2,294 4 letter words in my dictionary and 2,089 3 letter prefixes. The attack would function by trying all 6 letter words, then 5 letter words, then 4 letter words until a block is found that encrypts to the correct ciphertext. In total, to crack the first block (assuming my dictionary contains the words in the plaintext) this attack would require $6936 \times 26 + 4265 \times 248 + 2294 \times 2089 = 6,030,482$ calls to the oracle.

One issue was that the oracle provided is quite slow (requiring just over 3 seconds to perform an encryption), but I was able to get around this by calling multiple instances of the oracle in parallel. As I called more instances memory usage became a problem but I was still able to call 64 instances reliably, significantly increasing the speed of the program. Unfortunately, GPUs are unable to execute external exe files otherwise I would have been able to increase speed even more.

Although this attack is effective on paper, I was unable to run it for a significant length of time due to having to travel as a result of COVID restrictions. However, I was able to test the program with some simple cases (where the words are in the first hundred of the dictionary) and I feel confident that it would work within a few days.

References

[1] Giovanni Agosta et al. "Record setting software implementation of DES using CUDA". In: *2010 Seventh International Conference on Information Technology: New Generations*. IEEE. 2010, pp. 748–755.

[2] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.