CS6630 : Assignment 4 Report

Prasanna Bartakke EE19B106

16th September, 2022

Time-driven attack.

The time-driven attack relies on the cache hits/misses during the first round of encryption. The contents in the cache related to the T-tables are flushed, and the AES algorithm is executed for plaintext pt and key x. In the first round, 4 accesses are made to the T-table Te0 at indices corresponding to $pt[0] \oplus x[0], pt[4] \oplus x[4], pt[8] \oplus x[8], pt[12] \oplus x[12]$. During the first access of Te0($pt[0] \oplus x[0]$), 64 bytes of Te0 will be loaded into a cache line. During the second access, if the Te0 entry corresponding to $pt[4] \oplus x[4]$ is present in the cache, the time required for the execution of AES will be less than the case when it is not present in cache. Cache hits will happen when $pt[4] \oplus x[4]$ is in the vicinity of $pt[0] \oplus x[0]$. We denote this by $< pt[4] \oplus x[4] > = < pt[0] \oplus x[0] >$. Each cache line is of 64 bytes, and the size of each T-table is 1024 bytes. Thus, the T-tables are stored in 16 cache lines and the probability of cache hit is 1/16. In each iteration, we select a random plaintext and set the first 4 bytes of the plaintext to 0x0f. The cache is cleared and the time taken to run the AES algorithm is noted down. When the execution time of AES is less, we claim that this was due to a cache hit in the first round. From this we can come up with the key as follows:

We know there was a cache hit, so $< pt[4] \oplus x[4] > = < pt[0] \oplus x[0] >$. This can be simplified to $< x[0] \oplus x[4] > = < pt[4] \oplus pt[0] >$. Thus we can guess the first 4 bits of x[4]. Similarly, we get the first 4 bits of the remaining key bytes, x[5]-x[15]. Theoretically, we should be able to guess 12*4 = 48 bits of key accurately.

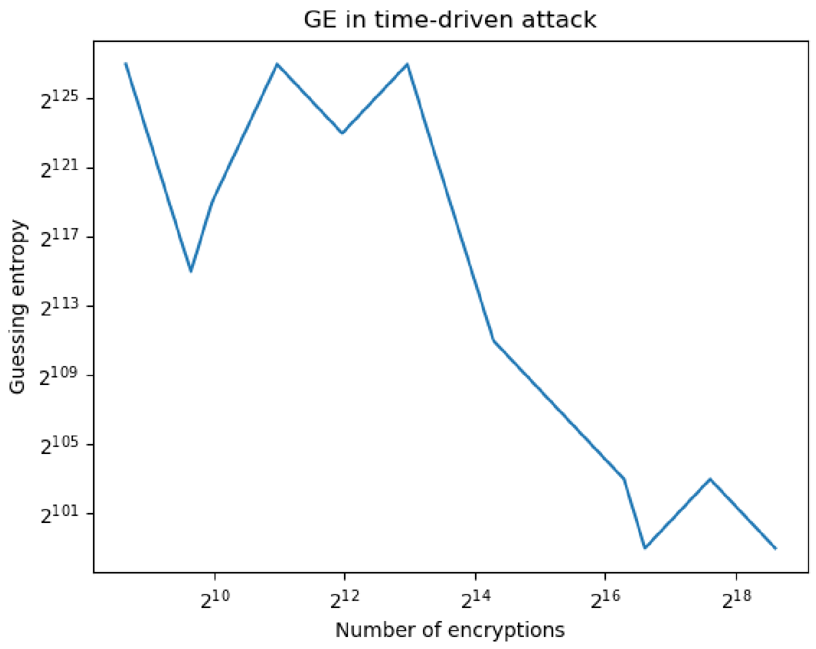
CPU details.

1. Number of CPU cores = 8.
2. L1 cache:
   (a) size = 32KB
   (b) associativity = 8 way associative
   (c) sets = 64

3. SMT threads: 2 hyperthreads per core. Processor (0, 4) share core 0. Similarly, processor (1, 5), (2, 6), (3, 7) share cores 1, 2 and 3 respectively.

Guessing entropy in time driven attack.

The guessing entropy is 2^{b-1}, where b is the number of bits needed to be guessed by the attacker to get the correct key after executing the attack. We can see that as the number of encryptions increases, the guessing entropy decreases. The following plot shows the guessing entropy vs number of encryptions in the time-driven attack.



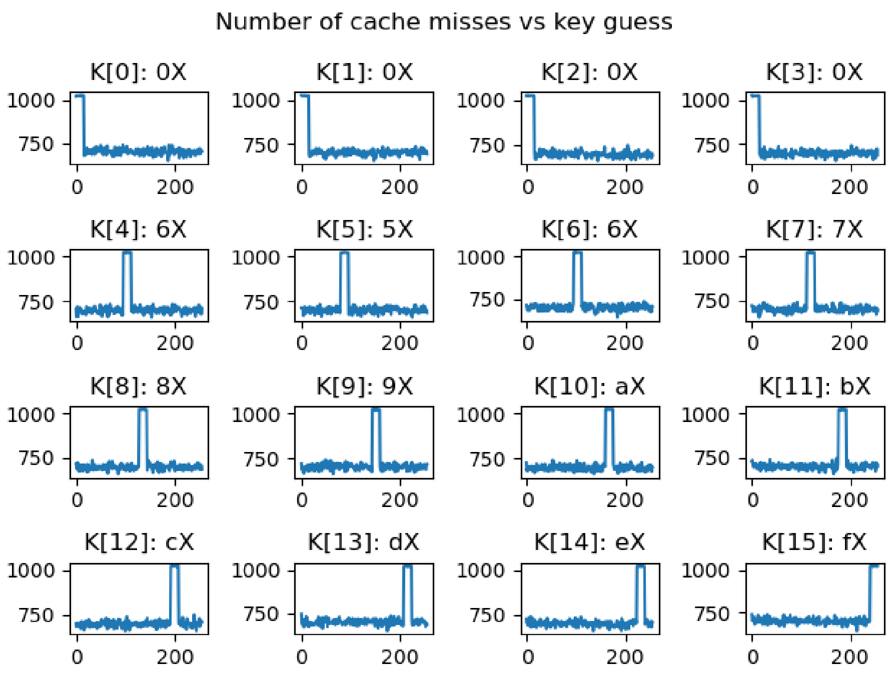
Using the time-driven attack we are able to recover only 24-28 bits of the key. Thus we are able to reduce the guessing entropy from 2^{127} to 2^{99}.

Evict+time attack.

In the evict+time attack we measure the time required for the encryption when the cache is filled with the required T-tables. Then the attacker manipulates the state of the cache at a specific location and observes the timing of the subsequent encryption. Depending on whether the time taken was significantly longer than the previous case or not, the attacker can conclude if the cache line that they evicted was accessed in the encryption and they can guess the key byte. For each byte position, we guess all possible key values from 0-255, and a random plaintext is chosen. Following are the steps for a given key byte(at byte number = pos, key guess = b) guess and a random plaintext(pt):

1. Set the first 4 bits of pt[pos] equal to the first 4 bits of b.
2. Remove the T-tables from the cache.
3. Encrypt the pt multiple times to warm up the cache. Measure the cache hit time of the encryption.
4. Evict the first block(index 0 - 15) of the T-table correspondng to pos from the cache.
5. Encrypt the pt and measure the evict time of the encryption.
6. If evict time - hit time > cache miss threshold, increment the miss frequency for pos and (b >> 4).

The above steps are done for multiple iterations. When will the evict time be significantly greater than the hit time? It will happen when the T-table block that we evicted from the cache was used in step 5. Suppose we are guessing the 6th(pos= 5) key byte(actual value = 50). We will iterate through all the possible key guesses from 0-255. When the first 4 bits of the key guess are 5, pt[5] is set to 5. During the T-table access in round 1, $Te1[pt[5] \oplus k[5]]$ is used. We can see that $pt[5] \oplus k[5]$ lies in the range [0-15], but Te1[0-15] was evicted from the cache by the attacker. Thus, the encryption will take longer time due to cache miss and there will be a clear distinguisher when the value of the first 4 bits of the key guess is 5. Similarly we can recover other bytes of the key. Theoretically we can recover 64 bits(4 bits from each byte) out of the 128 bits of the key. The following figure shows the cache misses vs key guess for the different bytes of the key. We can see that there is a peak when the first 4 bits of the key guess are equal to the actual key.

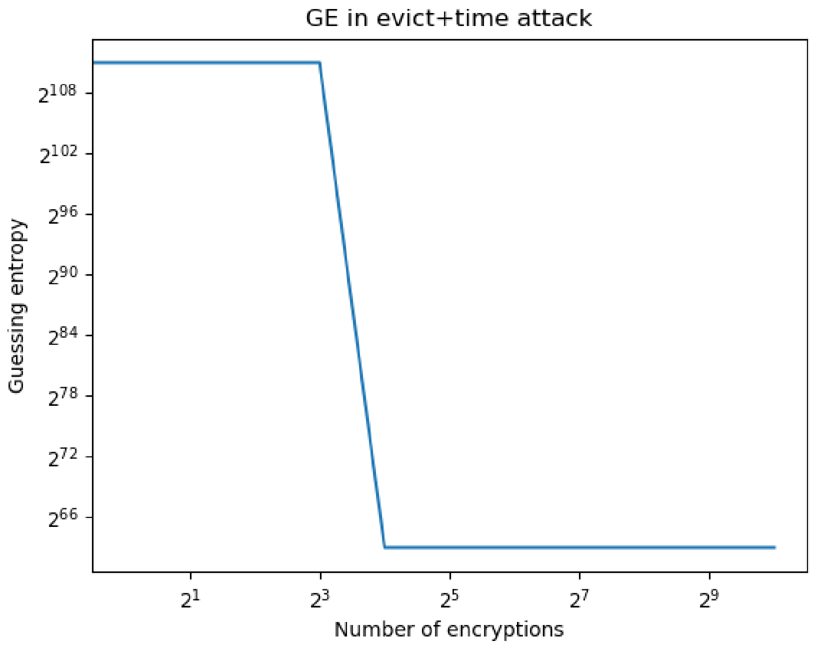


The file et_attack.c contains the code for the evict+time attack. Use the following commands to run the attack.

```
make clean
make et_attack
./et_attack
```

Guessing entropy in evict+time attack.

Using the evict+time attack, we are able to guess 64 bits of the key correctly. Thus we reduce the guessing entropy from 2^{127} to 2^{63}.


GE in evict+time attack
Guessing entropy
2^{108}
2^{102}
2^{96}
2^{90}
2^{84}
2^{78}
2^{72}
2^{66}
Number of encryptions
2^{1}
2^{3}
2^{5}
2^{7}
2^{9}