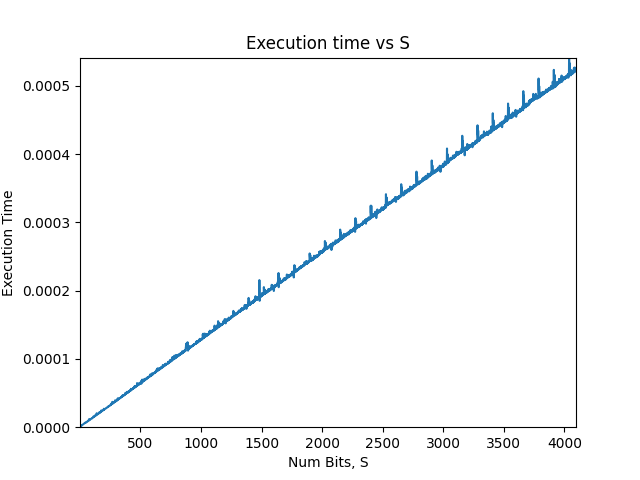
**HW1 048891**

**Winter 2024-2025**

**Adam Ghadban**

1. 

This graph was taken by choosing the following parameters:

a = 2

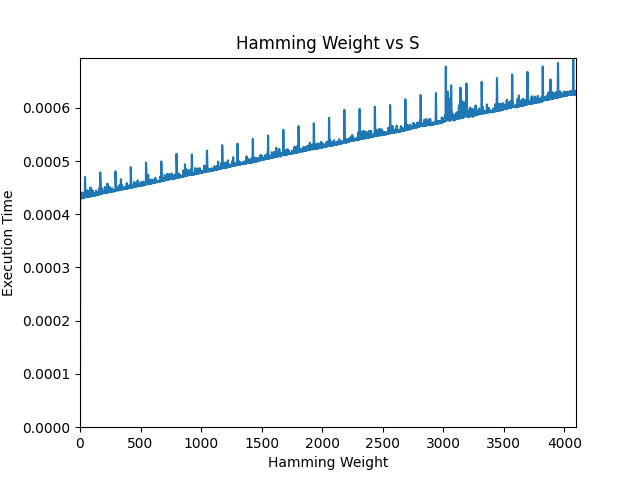
e = powers of 2 (1, 2, 4, 8, etc.. until 2 ^ 4096), which means s = [1, 2, 3, ..., 4097]

n = 20343797

The script was written using python.

Each S length was calculated 20 times then averaged over 20, to make sure we do not get spikes of computations and get as close to the average execution time as possible.

The graph state that the basic **left to write square and multiply** execution time is linear to the number of bits of e (s).

1. 

Here we also chose the following parameters:

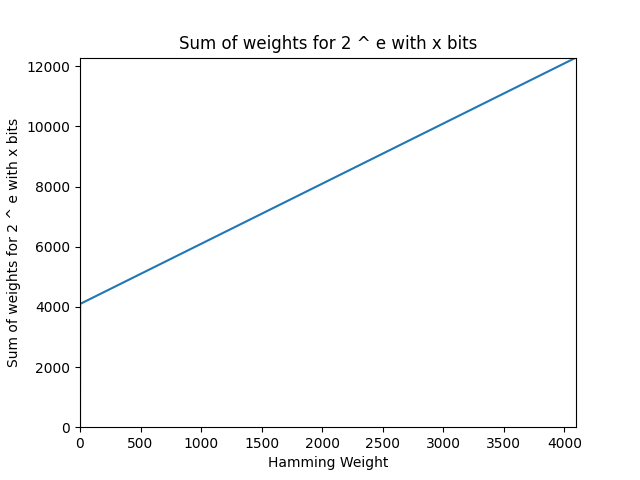
a = 2

e = [000..0, 1000000.., 110000.., 111000.., .., 11111...110, 1111...111]

n = 20343797

We can see a linear behavior, with the exceptions of spikes due to memory and caching (most likely).

This means that with every increasing hamming weight (another bit equal to 1), we add more operations. This shows that we can use side-channel to attack this algorithm to figure out the exponent.

1. 

We set a weight of 1 for the exponentiation, and a weight of 2 for the multiplication, and the graph shows the sum of the weights for every operation of 2 ^ e, where e is an exponent with 4096 bits with a certain hamming weight.

For an exponent with hamming weight = 0, we expect a weight of 4096, and with each big going from 0 to 1, we add another 2, since we want to square, but also want to multiply, which results overall in a gradient = 2, and an end weight sum of 4096 \* 3 = 12288.

This suggests that the number of operations differ per exponent, as we learned in class.