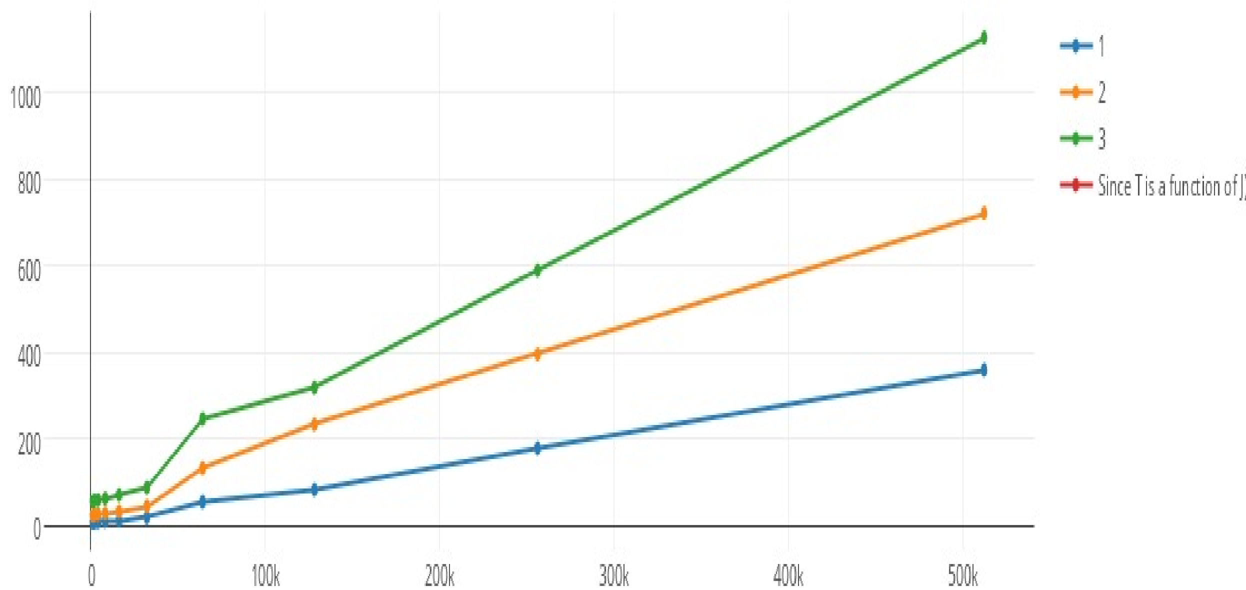
Analysis of ACC:

Methodology:
We plot the average time taken per authentication access against M.
We sample M using equation m1= 2*m0.
Sincce T = j*l, and l essentially remains unchanged for our system, we use J to group the line-plots.



Using the above graph as a validation, we can say that time to authenticate increases linearly with both the M as well as T. However, the plot has a steeper slope with respect to T.
One must note however that the equations only consider a Random Access to memory. On a larger system with disk-based access, the disk-read cost would have a significant impact as well.

Upon my system with 1Gb of RAM, I was able to work with M = 512k, safely upon my system.
In this scenario the ram consumed would be = 512k*128 bits = 100 Mbytes on an average, beyond which the implementation is to be blamed.
Theoritically we can have as many as 2^max_bitlengths. However, it is often constrained by various system and other usages and the actual usage space lesser than 4Gb is yet large enough.

Analysis:
While computation-time for Acc should ideally be linear with |m|, since, we use a constant bit-length for m, the relationship essentially behaves linear with respect to time. To account for the fact that dividing by larger numbers would take longer time, the linearity comes in force.
However, a J has a direct impact on the time taken to compute as a higher value of j implies j*l permutations of R(t). Each of this permutations, implies an xor at a larger state. Thus while the look inexpensive, their children do bring a multiplied factor of work.