Technique Specifics

Before encryption, the user has to provide a password which is stored into a vector (per word as an entry) . The password is then expanded by a large, random-generated matrix into a 5000 by 1 vector; this vector is the *key* to this encrypting process. The large vector is then divided into several sub-vectors which are all 10 by 1. Those columns vectors are put into a blank matrix one by one such that ensuing vectors and the existing vectors (vectors in the matrix) are linearly independent. If a vector is to be put into the matrix is linearly dependent to the existing vectors, then it will be discarded. Let's call the constructed matrix A, and matrix $E = A^T A$. Now E is an *invertible* matrix for A has *linearly independent columns*, which means it can be used to perform encryption, and decryption -- the reverse of encryption.

After generating an invertible matrix, we multiply the matrix with our target image, and the image becomes "unrecognizable", which means it has been successfully encrypted. To decrypt it, we multiply the image by the inverse of the generated matrix.

Since an image can maximally store `uint8` values, the large integer after the above matrix multiplications must be handled with modulus. And there is another problem: the entries of E^{-1} would be quite small if we directly use `numpy.linalg.inv`, since it has been divided by $det(E)$. This result is not agreeable for we don't want to deal with floating-point numbers, which is much more tricky to handle. As a result, we calculate the *modular multiplicative inverse* of $det(A)$, and multiply it to our image directly; this would work perfectly well since the the *modular multiplicative inverse* of $det(A)$ would at most be *256*. Why *256* ? This is another story.

We know that for $aa^{-1} \equiv 1 \pmod{p}$, a^{-1} exists if and only if $gcd(a, p) = 1$. Unfortunately, the maximum of `uint8` is *256*, which has *2* as its factor, and hence the odds to successfully find modular multiplicative inverse is quite low. Then it comes ***257*** -- a prime number. With ***257*** being a prime number and its trivial difference from *256*, we adopt it to $\mod$ our matrices; we also try to eliminate the difference by adding and subtracting 1 after and before $\mod$.