28. (*expires 4/10*) A Vignère cipher can be interpreted as a Caesar-like cipher on n-vectors, where n is the length of the key phrase. Can every affine encipherment on digraphs (two-character codes) be interpreted as an affine matrix encipherment on 2-vectors? That is, suppose I encode a message by affine enciphering on digraphs. Can I always get the same crypttext from the same plaintext using an affine matrix enciphering (using a 2×2 matrix) on 2-vectors?
If your answer is yes, prove it. If no, give a counterexample that cannot be so interpreted.

29. (*expires 4/10*) It was mentioned in class that the implementation of the `AffineMat` cipher (see Crypto.mw) could be adjusted to that if the shift vector b is omitted, the zero vector is used instead. Write a version of `AffineMat` that does this. More specifically, change the code so that `AffineMat(text,A)` works exactly the same as `AffineMat(text,A,b)` where `b` is the appropriate zero vector. Keep in mind that `A` may be of any size.

30. (*expires 4/10*) The difficulty of breaking a cipher can be increased by inserting some random characters (or noise) into a known part of the plain text in such a way that it will interact with the actual text (sometimes this insertion of randomness is called "salting the plaintext").

    As an example of this, recall that we discussed that if our character set came from an alphabet of length n, we could represent blocks of k-characters as numbers base n^k (for example, in the 53-character alphabet consisting of a space, upper-case letters and lower-case letters, the word `Hi` is represented by $8 + 35 \cdot 53 = 1863$ if we use 2-character blocks).

    If we agree that the first character of each block will be randomly chosen (and just ignored when we decrypt), breaking the encryption becomes much, much harder.

    Modify the encryption scheme in `BigAffine` from Crypto.mw so that it can encrypt and decrypt including salt as the first character of each block.

    As an explicit example, we encode the string `Zombie Apocalypse` using the 53-character alphabet and 3-character blocks (where the first character of each block is random noise). The given string corresponds to the list of character codes

    $$[26, 41, 39, 28, 35, 31, 0, 1, 42, 41, 29, 27, 38, 51, 42, 45, 31]$$

    which, when grouped into pairs and with a random character added as salt, gives me

    $$[116551, 80721, 88936, 2842, 117404, 77428, 145275, 128676, 1652]$$

    (your numbers may differ by up to 53 because of the salt); viewing these as 3-graphs including the salt, corresponds to `DZoBmbBieg AIpovcaBlyspsIe`. If we encrypt this using the affine encryption $x \mapsto 12347x + 56890 \mod 53^3$, we get

    $$[67005, 136439, 32930, 12092, 28729, 121189, 97219, 4118, 57985]$$

    or `MsWQdvQlKHPDCLJeGqQfhkXAChT` (unless you used the same salt, your encryption will differ significantly, but both should decrypt just fine.)

    If you were able to make this work, you should be able to decrypt the string[1]

    `MWLiuWVckaMOpfHuWPgGuWlyQovqkwBwWV nLZYhrySYihTUSFqFJuGxEtvxCWNxPxstkPwkkxo`

    which was encrypted using salted pairs (that is, 3-character blocks including salt) from the 53-character alphabet above, and applying the affine mapping $x \mapsto 47x + 31415 \mod 53^3$.

[1] also available as zombiecrypt.txt so you don't have to worry about typing errors.