CSC 440
HW1
Ximan Liu

1a)
Encryption formula: y = 19x + 12
Plaintext: pandemic
Ciphertext: **LMZRKGIY**

Python code:
```
def affine_encrypt(text, key):
    return ''.join([ chr((( key[0]*(ord(t) - ord('A')) + key[1] ) % 26)
                  + ord('A')) for t in text.upper().replace(' ', '') ])

def main():
    # declare text and key
    text = 'pandemic'
    key = [19, 12]
    # call encryption function
    affine_encrypted_text = affine_encrypt(text, key)
    print('Encrypted Text: {}'.format( affine_encrypted_text ))
if __name__ == '__main__':
    main()
```

1b)
Decryption formula: **x = 1/19 * (y - 12) = 11 * (y + 14)**
Decryption function:
```
# Extend Euclidean Algorithm for finding mod inverse
def egcd(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    gcd = b
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = egcd(a, m)
    if gcd != 1:
        return None
    else:
        return x % m
```

```
# affine cipher decryption function
# return original text
def affine_decrypt(cipher, key):
    return ''.join([ chr((( modinv(key[0], 26)*(ord(c) - ord('A') - key[1]))
                    % 26) + ord('A')) for c in cipher ])

# test
def main():
    # declare text and key
    affine_encrypted_text = 'LMZRKGIY'
    key = [19, 12]
    # call decryption function
    print('Decrypted Text: {}'.format
    ( affine_decrypt(affine_encrypted_text, key) ))
if __name__ == '__main__':
    main()
```

1c)
Ciphertext: VMYYIZK
Plaintext: **VACCINE**

2)
Use inverse of a and b to decode affine cipher and encrypt the plaintext. In that case we should find numbers are co-prime. Total size of key space is **33 * 20 = 660**.

3)
31 * 30 = 930.

4)
(7) h -> N (13) $7\alpha + \beta \equiv 13 (\text{mod } 26)$
(0) a -> O (14) $0\alpha + \beta \equiv 14 (\text{mod } 26)$
$13 = 7\alpha + 14 (\text{mod } 26) \Rightarrow 7\alpha \equiv -1 \equiv 25 (\text{mod } 26)$
$\alpha = 11$
$0(11) + \beta \equiv 14 (\text{mod } 26)$
$\beta = 14$
$y = 11x + 14 (\text{mod } 26)$

5)

Keylength.java

Output:

1, 26
2, 23
3, 17
4, 22
5, 20
6, 29
7, 28
8, 26
9, 19
10, 37
11, 21
12, 22
13, 19
14, 29

```
public class keylength {
	public static void main(String[] args) {
		String ciphertext = 
"XVMRNEBXPAEWBZQCDLMOEGSVBYFPAWKKZRSDMKSLOMAKKAAFDWZMZVFPDFWLPMKRO
WBUDHYYZZWUWDAOEGIKLKBVGHNCBZPAMFFYEWQKDSSSDLPIBYDPPVRLJWEZWDVKESOXPV
DQHYPQPMKRWECELQZIVTUUDKTRJMVTOQRZYJAKXIRPSTELJKGFXNQIPGEXKRUZBLXEAVPRSLS
EFVERORFNMGEROCAKHIPRDHZSVWDYHFCLJKQEWAUVCVKVLZVYFVEHHSORUEHYXVKVIWHSC
KSHNVMCDPSUAUKFTVPOWEYXIFMIWDSFCBWJCCOQBZGHNWICTQOEEXIGWDSQHVCLBFCZOP
WJVQKAVKRXSRMOAXWSUAOBLOHSNKKEGYLYEROJDERKSDPAMROHYEZZPSLRBPVREKWGSVU
OOEOLJXMCOEUVYFAEOVQYWVDFWRKFPLFFXLOIXVRLZVDGWXIZQDIEOUAHAFIICIPSNSARLYK
RJVPLIEEUPLTOZMVXDMIRYWQQKFPLIKPUQWCROHMKSHUHWEWAJVYEKXPVUPFPTQCXWSUA
OBEKAIVTUUDKTRJVMCBEBXTQOXMRGKBGZRNMUGOAAVYWWXQFQOOEOKQQIEHNFFC";
		for (int shift = 1; shift <= 14; shift++) {
			int coincidenceCount = 0;
			for (int index = 0; index <= ciphertext.length() - 1; index++) {
				int shiftedIndex = (index + shift) % ciphertext.length();
				if (ciphertext.charAt(index) == ciphertext.charAt(shiftedIndex)) {
					coincidenceCount++;
				}
			}
			System.out.println(shift + ", " +  coincidenceCount);
		}
	}
}
```

6)

Key: weirdworld

Breaking existing public key cryptography. This is the most direct political and security implication. Everytime you visit a website that begins with https the authentication and encryption, including e.g. protecting your credit card number happen using a cryptosystem. Based on factoring integers or discrete logarithms, or a few other related problems, in number theory a full universal quantum computer, if built is known to be able to break all of this. Having said that we all know today that, hackers and intelligence agencies can compromise people's data in hundreds of more prosaic ways than by building a quantum computer. Usually, they don't even bother trying to break the encryption relying instead on poor implementations and human error.