Cryptography – Individual Report

The Affine Cipher –
An Analysis of the Strengths and Weaknesses

Abstract:

The affine cipher is a monoalphabetic shift cipher derived from historic ancestor, the Caesar cipher. It consists of using mathematics on a multiplication basis to produce cipher text. In addition to this, the report looks at the simplicity of the cipher and how easy it is for it to lead to brute force. The affine cipher, albeit popular, has some security drawbacks which will be discussed in this paper with possible solutions for securing the cipher against its own drawbacks. I will look at how secure it is, the mathematics involved, and other methodologies linked with this cipher. After concluding an analysis via creation of an affine cipher brute force, encryption and decryption tool in python, this report will critically look at all aspects and base a conclusion on these findings.

Word Count (Minus Headings & Bibliography): 1072

Index

Introduction:

There are two main types of Cryptosystems, symmetric and asymmetric; with symmetric being that the same key is used to encrypt and decrypt. With asymmetric, a public key is used to encrypt a message and then a private key is used to decrypt the message.

The Affine Cipher Mathematics:

The affine cipher provides a mathematical approach to a monoalphabetic shift cipher. Relying heavily on the modulo m, by which to say, the length of the space chosen. For example, the English alphabet of 26 letters; this would be represented by **mod26**. [1]

For encryption, we must change each of the letters in the plaintext to the corresponding number using 0 to 25 (Fig.1.) The encryption formula in mod26 is as follows:

Encryption: $Ek=(\propto p+\beta) \bmod 26$. ***(∝ is alpha & is β beta).***

The affine cipher works on a multiplicative basis as we have to multiply the number corresponding to the plaintext letter by alpha then add b to the total. Alpha must be such that alpha and modulo are coprime.

For decryption we are basically doing the opposite, the technical term is 'multiplicative inverse' and is where you would utilise the extended Euclidean algorithm mathematical technique. It is important to note that with this process, we start the same as encryption, assigning all of the letters to the corresponding numbers (as shown in fig.1.). We must then find the greatest common divisor (**gcd**). The decryption formula is as follows:

Decryption: $Dk=\propto -1(C+\beta -1) \bmod 26$

Assuming that c is the mod multiplicative inverse (fig2.) of alpha (***∝)*** and must always be equal to 1 once the full calculation has been completed such that

$\propto x\ c=1.$

To understand more thoroughly; when you multiply alpha by c and deduct the modulo, you get the answer of 1. The formula for finding the multiplicative inverse is: ***x=mod(x,m).***

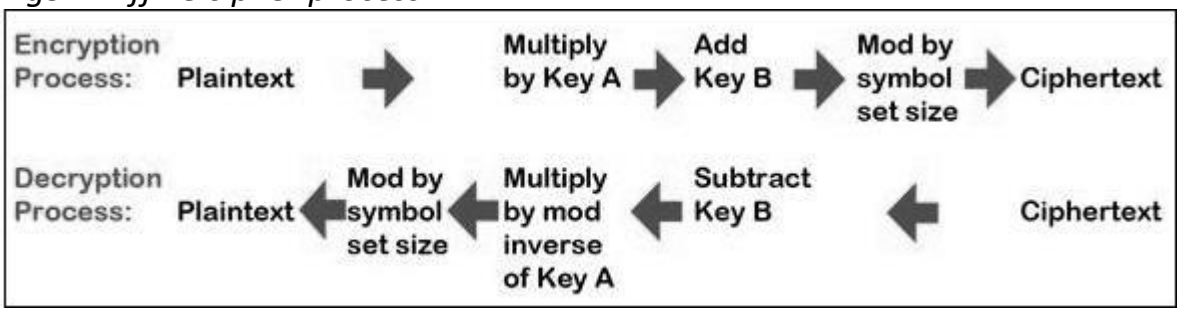
Fig1. Plaintext alphabet and value [1]:

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Plaintext Value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Fig2. – Inverses on mod26

Inverses mod 26												
x	1	3	5	7	9	11	15	17	19	21	23	25
x^{-1}	1	9	21	15	3	19	7	23	11	5	17	25

Fig3. – Affine cipher process



Example [8]:

Plaintext = HELLO > Mapped to: 7,4,11,11,14
Function = f(∝)=3∝+7
Output Post-Function f(∝) = 2,19,14,14,23
Cipher Text Output = CTOOX
NB: You can see here that there is an index of coincidence with the 2 o's.

Cracking the ciphertext:

We were given an encrypted message to crack using a programming language of our choice while demonstrating the Affine Cipher and the extended Euclidean algorithm. We used python as our code of choice to brute force the cipher text, and as an additive we have ensured that our brute force code can crack any cipher text inputted by the user.

Encrypted text provided to us: GRKTPUCRITBKPERWAVUXUCK

We were aware that it was encrypted using the affine cipher, so checked if there would be an index of coincidence [5]. We noted that the letters ***R, K & U*** appears three times each. On this basis we worked backwards using the mathematical techniques described to clarify the key and result. The formula to calculate the index of coincidence is [6]:

Fig.4. Showing the formula for the index of coincidence [6]

$$\text{Index of coincidence for random text} = \frac{26 \times \frac{Tt}{26}\left(\frac{Tt}{26} - 1\right)}{Tt(Tt-1)}$$

We learned the following:
R = r and K = y and U = o
The key was discovered to be: a=25, b=8
The decrypted phrase is: CRYPTOGRAPHY TERMINOLOGY

The Brute Force Code:

Fig4. – Our Code - Functions

```
from math import log # imports

def egcd(a, b): # extended euclidan fuction
  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): # Finding the inverse of the first key (GCD)
  gcd, x, y = egcd(a, m)
  if gcd != 1:
    return None # modular inverse does not exist
  else:
    return x % m

def infer_spaces(s): # function to split a string into words
    words = open("dictionary.txt", encoding="latin-1").read().split()
    wordcost = dict((k, log((i+1)*log(len(words)))) for i,k in enumerate(words))
    maxword = max(len(x) for x in words)

    def best_match(i):
        candidates = enumerate(reversed(cost[max(0, i-maxword):i]))
        return min((c + wordcost.get(s[i-k-1:i], 9e999), k+1) for k,c in candidates)

    # Build the cost array.
    cost = [0]
    for i in range(1,len(s)+1):
        c,k = best_match(i)
        cost.append(c)

    # Backtrack to recover the minimal-cost string.
    out = []
    i = len(s)
    while i>0:
        c,k = best_match(i)
        assert c == cost[i]
        out.append(s[i-k:i])
        i -= k

    return " ".join(reversed(out))


def convert(lst): # function convert a sentance into a array
    return ([i for item in lst for i in item.split()])
```

Fig.4. Continues on next page

Fig4. – Our Code – Main Code

```
# Driver Code
def main():
          text = input("please enter the cipher text: ")
          key1 = [1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25] # key value 1 (A) (12 possiblitys) has to be co-prime with the mod
          key2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] # key value 2 (26 possibilitys)
          i = 0
          j = 0
          key1len=len(key1)
          key2len=len(key2)
          possibilitys=[]
          plaintexts = []
          sentance = []
          for i in range(key2len): # for loop of key B
              g = key2[i]
              for j in range(key1len): # for loops of key A
                 f = key1[j]
                 j = j+1
                 key = [f, g]
                 fstr = int(f)
                 gstr = int(g)
                 decrypt =  (''.join([ chr((( modinv(key[0], 26)*(ord(c) - ord('A') - key[1])) % 26) + ord('A')) for c in text ])) # code to decrypt

                 long_string = decrypt.lower() # converst the decrypted text to lowercase
                 possiblewords = (infer_spaces(long_string)) # function to find the words in a long string
                 lst =  [possiblewords] # adds that to a array
                 splitlist = convert(lst) # splits the words into an array
                 matches = [] # declares an array

                 for o in splitlist:# for each word
                   #print (o)
                   words = open("dictionary.txt", encoding="latin-1").read().split() # opens files
                   if o in words: # if word is in the dictionary
                     matches.append(o) # add word to array
                   if matches == splitlist: # if the arrays match ie everyword is in the dictionary add to possible arrays
                    possibilitys.append(key) # adds the key to the array
                    plaintexts.append(decrypt) # adds the plaintext to the array
                    sentance.append(possiblewords) # adds the sentance to the array

              i=i+1

          print ("The possible keys are ", possibilitys) # prints the keys
          print ("the possible plaintext are ", plaintexts) # prints the plaintext
          print ("the possible sentances are ", sentance) # prints the sentance


if __name__ == '__main__':
  main()
```

Our Code Explained:

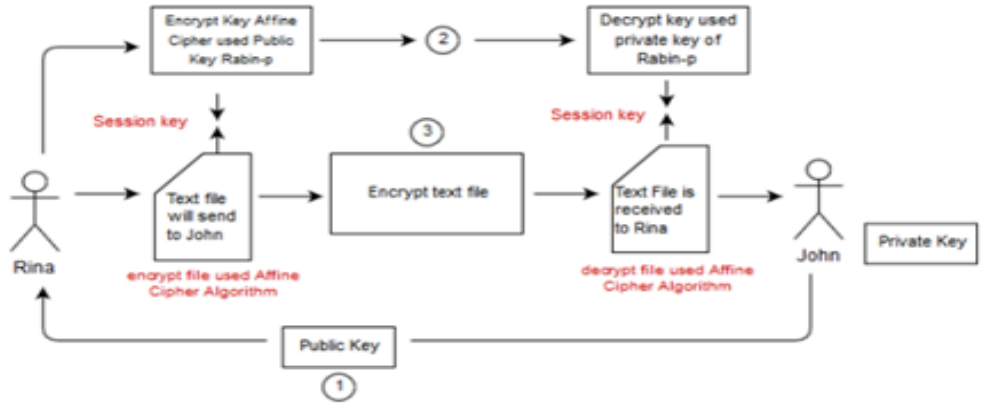
At the start you will be asked to enter in your cipher text. The program will then begin and set the first possible keys (a and b), at this point it will find the modular inverse of these keys and runs the decrypt function ($P = (a^{-1} * (C - b)) \%26$) It will then convert the text to lowercase and run the infer_spaces function to check the most likely spacing of the plaintext. It will then compare the text to the dictionary and checks if there are any viable matches; at this point the program will loop until it has the likeliest match and print out the answer along with the keys discovered.

Exploring Potential Security Issues:

The affine cipher has 312 possibilities and is vulnerable to known-plaintext-attacks. [3] [M A Budiman et al] suggests combining a second method with the affine algorithm to secure it. In this case they have chosen a hybrid methodology and combined asymmetric and symmetric algorithms(see fig.5.). [2]

In 2011 the affine-Hill Cipher was introduced by Tourani and Falahati, built on the foundations of the affine cipher, inadvertently created a cipher that was found to be vulnerable to zero-plaintext-attacks due to an issue with the protocol. [9] proposes combining the affine-hill cipher with Arnold Transform (chaotic map methodology) to fix the protocol creating the weakness with the affine-hill cipher and claims limited time constraints.

Fig.5. Flowchart of proposed Affine cipher security improvement [2]



Conclusion:

The affine algorithm can be made more secure in various ways, including the affine-hill cipher and many proposed upgrades to make it more secure, in terms of encryption, there are more secure systems out there to utilise and prevent attacks. The Advanced Encryption Standard(AES), a symmetric cipher, replaced Data Encryption Standard(DES) in 2001 due to DES becoming increasingly vulnerable to brute-force attacks, AES is now the globally recognised standard of encryption and was developed by The National institute of Standards and Technology(NIST). It is also worth mentioning RSA(Rivest-Shamir-Adleman), an asymmetric cipher, which is frequently used in the encryption of online data. AES is considered secure against brute force attacks and will only become insecure once quantum computing becomes available to threat actors [7]. When considering Kerkhoffs's principle, the affine cipher does not withstand secureness'; therefore, should not be considered a viable modern day cryptosystem [4].

Recommendations:

I would not recommend using the affine cipher as a stand-alone algorithm in any setting unless combined with another asymmetric algorithm, but it is worth mentioning there are far better choices for encryption and decryption to choose from to better enhance security of any company's data. I would not recommend using the affine cipher in a technical setting that needs robust encryption ciphers to protect data unless using the Affine Transformation(AFT) for encrypting images.

Bibliography:

[1]D. Rodriguez-Clark, “Affine Cipher,” *Crypto Corner*, 2013. https://crypto.interactive-maths.com/affine-cipher.html (accessed Mar. 17, 2022).

[2]M. A. Budiman, Handrizal, and S. Azzahra, “An implementation of Rabin-p cryptosystem and affine cipher in a hybrid scheme to secure text,” *Journal of Physics: Conference Series*, vol. 1898, no. 1, p. 012042, Jun. 2021, doi: 10.1088/1742-6596/1898/1/012042.

[3]Wikipedia, “Affine cipher,” *Wikipedia*, Feb. 06, 2020. https://en.wikipedia.org/wiki/Affine_cipher.

[4]Wikipedia Contributors, “Kerckhoffs’s principle,” *Wikipedia*, Dec. 17, 2019. https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle.

[5]Cryptocrack, “CryptoCrack - Index of Coincidence,” *sites.google.com*, 2013. https://sites.google.com/site/cryptocrackprogram/user-guide/statistics-tab/index-of-coincidence (accessed Mar. 18, 2022).

[6]Ciphertools, “Index Of Coincidence Formula Image,” *Ciphertools.co.uk*, 2022. https://ciphertools.co.uk/Images/IndexOfCoincidenceFormulaRandom2.jpg (accessed Mar. 18, 2022).

[7]C. Bernstein and M. Cobb, “What is the Advanced Encryption Standard (AES)? Definition from SearchSecurity,” *SearchSecurity*, Sep. 2021. https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard.

[8]Emory Oxford College, “Breaking an Affine Cipher,” *mathcenter.oxford.emory.edu*, 2013. http://mathcenter.oxford.emory.edu/site/math125/breakingAffineCiphers/ (accessed Mar. 18, 2022).

[9]M. Hadi Valizadeh, “Healing the Hill Cipher, Improved Approach to Secure Modified Hill against Zero-plaintext Attack,” Eprint, 2016. [Online]. Available: https://eprint.iacr.org/2016/806.pdf.