INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Cryptography

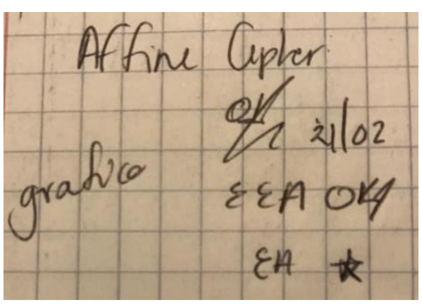
Affine Cipher

Classic cryptography: Affine cipher program

By:
Ana Paola Nava Vivas

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

March 2017



Index

Contenido

1. Introduction:

In this report I’ll explain how I did the program for Affine cipher.
Since a shift cipher can produce only 26 different transformations for the text in the English alphabet, it’s not a very secure method to cipher text. Affine encryption instead, is a generalization of the encryption of change that provides a little more security.
We’ve worked on this in class by doing the shift cipher program first and modifying it step by step.
In its final form, the program must do the transformation of the text at an interface.
For example: the word “cryptography” would be transformed into “FUBSWRJUDSKB” with $\alpha=3$ and $\beta=0$.
The ways to achieve that are explained with more detail at the procedure, and the theoretical part at the literature review.

2. Literature review:

At first we were introduced to the *shift cipher*. It is actually a special case of the substitution cipher in which we simply shift every plaintext letter by a fixed number of positions in the alphabet. For instance, if we shift by 3 positions, A would be substituted by d, B by e, etc. At the end of the alphabet, X, Y and Z just “wrap around”. That means X should become a, Y should become b, and Z is replaced by c. [1]

Then, we tried to improve the shift cipher by generalizing the encryption function. The affine cipher encrypts by multiplying the plaintext by one part of the key followed by the addition of another part of the key. [1]

Definition 1.4.4 Affine Cipher
Let $x, y, a, b \in \mathbb{Z}_{26}$
Encryption: $e_k(x) = y \equiv a \cdot x + b \bmod 26.$
Decryption: $d_k(y) = x \equiv a^{-1} \cdot (y - b) \bmod 26.$
with the key: $k = (a, b)$*, which has the restriction:* $\gcd(a, 26) = 1.$

Figure 2.1. (Definition for the Affine cipher)

The decryption is easily derived from the encryption function:

$$
\begin{aligned}
a \cdot x + b &\equiv y \bmod 26 \\
a \cdot x &\equiv (y - b) \bmod 26 \\
x &\equiv a^{-1} \cdot (y - b) \bmod 26
\end{aligned}
$$

The restriction gcd(a ,26) = 1 stems from the fact that the key parameter a needs to be inverted for decryption. We recall from that an element a and the modulus must be relatively prime for the inverse of a to exist. Thus, a must be in the set:

$$a \in \{1,3,5,7,9,11,15,17,19,21,23,25\}$$

But how do we find a^{-1}? For now, we can simply compute it by trial and error: For a given a we simply try all possible values a^{-1} until we obtain [1]:

$a \cdot a^{-1} \equiv 1 \bmod 26$

For instance, if $a = 3$, then $a^{-1} = 9$ since $3 \cdot 9 = 27 \equiv 1 \bmod 26$. Note that a^{-1} also always fulfills the condition gcd(a^{-1},26) = 1 since the inverse of a^{-1} always exists. In fact, the inverse of a^{-1} is a itself [1].

For example: let the key be k = (a,b) = (2,3), and the plaintext be HMZIPGOMDIQZ, the inverse a^{-1} of a exists and is given by a^{-1} =9 . The ciphertext is computed as: cryptography.

As we progressed in class we learned the Euclidean extended algorithm, which allows us to decrypt ciphered texts that used bigger keys to be encrypt in a recursive way.

3. Software (libraries, packages, tools):

Libraries: Stdio.h (for my C first version), W3.css framework for the html.

Packages: none.

Tools: I used Atom text editor as a tool to edit html and the python idle.

4. Procedure:

1- At first, I made a program that could change text from lowercase to uppercase and vice versa from my m.txt file to c.txt file. In C language and many others, it is possible to just read a letter as its ascii number, so the transformation from lowercase to uppercase was a subtraction, and the transformation from uppercase to lowercase was a sum by 32.

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figure 4.1 (ASCII chart table)

2- Then we introduced a key and a menu, so we could choose a number to also shift the letter. At this point it was a simple Shift Cipher (or Caesar Cipher).

3- After doing the Shift Cipher we introduced the α number which allowed us to make more complex this encryption and difficult to decrypt, also, the key for the shift was transformed more formally into a β.

4- At this point the program was ready but we could make it better by using the Euclidean extended algorithm instead of the brute force.

I transcript my program to python in which I already had the following Euclidean function.

```
def euclidex(a,b):
    if(a%b==0):
        return 0,1,b
    else:
        m,n,r = euclidex(b,a%b)
        return n,m-(a//b)*n,r
```

Figure 4.2 (python function for the Euclidean extended function).

```
function euclidex(a,b){
  if(a%b==0){
    return [0,1,b];
  }
  else{
    var temp=euclidex(b,a%b);
    m=temp[0];
    n=temp[1];
    r=temp[2];
    return [n,m-parseInt(a/b)*n,r];
  }
}
```

Figure 4.3 (same function but in JavaScript).

5. Results

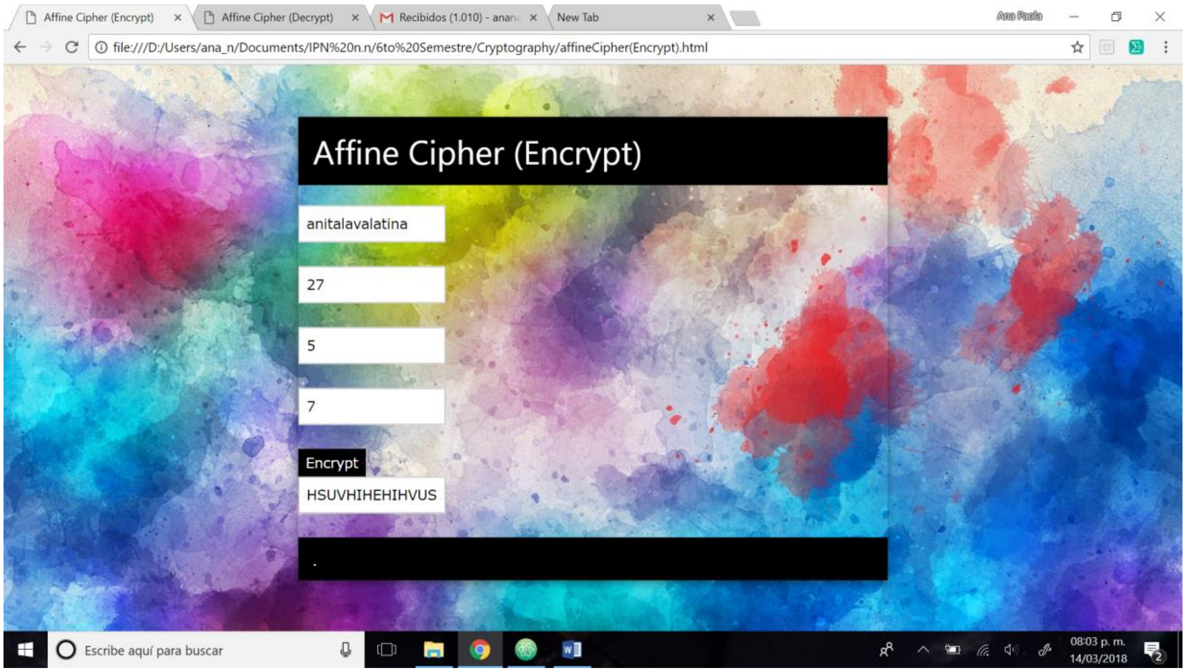


Figure 5.1 (Result for the encryption of "anitalavalatina")

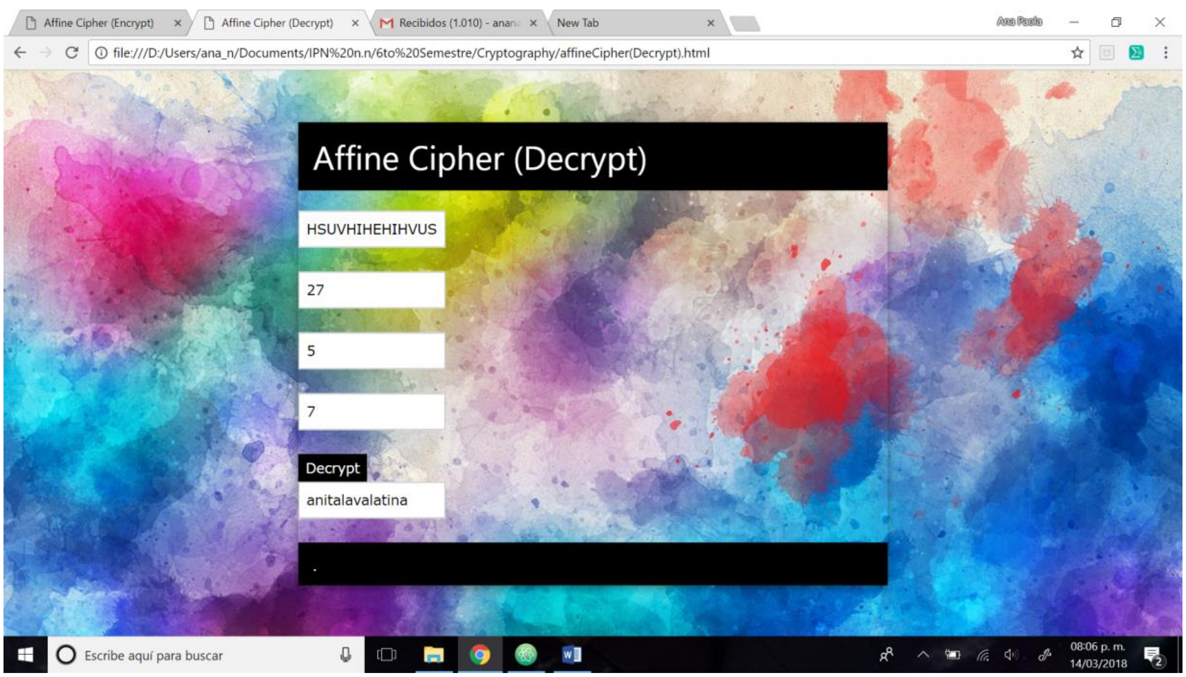


Figure 5.2 (Result for the decryption of "HSUVHIHEHIHVUSH")

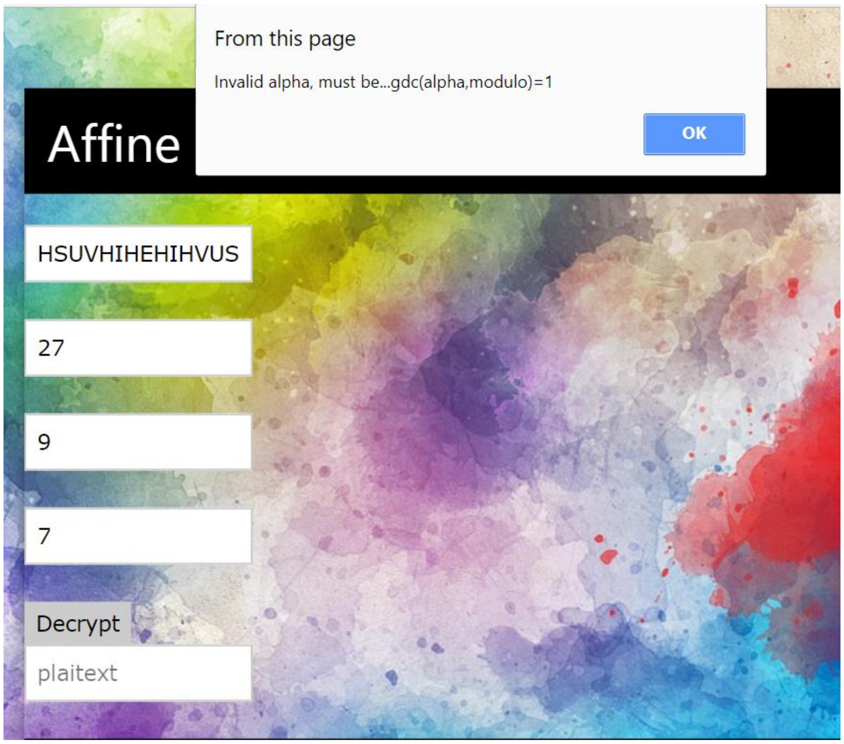


Figure 5.3 (VALIDATION: an alert when the gcd(alpha,modulo) is bigger than 1)

6. Discussion:

The encryption interface shows 4 inputs, one for the message, one for the modulo, one for the alpha, and the beta. When the “Encrypt” button is pressed, a ciphertext is showed. The following equation was used to cipher the original message:

$$C=\alpha*m + \beta$$

m: represents a letter in the original message.
C: represents the substitute for m.
β: represents a simple shift.
α: represents a number which function is basically complicate the ciphertext.

The same applies for the decryption interface, but it’s a bit different. Here it was important to use something called “Euclidean expanded algorithm” to find the multiplicative inverse of alpha.

$$m = (C + (-\beta)) * \alpha^{-1}$$

A validation was needed, where it was important to determine if the gcd for α and the modulo was greater than 1. I did that through the Euclidean function, because the third value of the returned array was gcd(α,modulo), so it was useful for just comparing that to 1, and if it was greater, an alert would be showed preventing the user from calculating something wrong.

7. Conclusions:

Maybe nowadays it's a bit easy to break the affine cipher with a desktop computer, but in the past, there weren't computers that make it easy to use the brute force to break a code, plus not everybody knew how to read, so it used to be a very efficient ciphering method.

Today, more than efficient, is a good way for being introduced at cryptography. Sadly, it's hard to think of a real life situation to use this method since it's not a very secure encryption. I'd use it with friends for unimportant messages or games.

8. References:

[1] Paar, C. and Pelzl, J. (2010). Understanding Cryptography. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

9. Code

This is the solution in python.

```python
def euclidex(a,b):  #this function is recycled from “analisis de algoritmos”
    if(a%b==0):
        return 0,1,b
    else:
        m,n,r = euclidex(b,a%b)
        return n,m-(a//b)*n,r

def lectura(op,mod,a,b):
    if(op==1):
        arch=open("m.txt","r")
        archo=open("c.txt","w")
        archo.write("")
        archo.close()
    elif(op==2):
        arch=open("c.txt","r")
        archo=open("m.txt","w")
        archo.write("")
        archo.close()
    message=arch.read()
    for i in message:
        if(op==1):
            c=(((ord(i)-97)*a+b)%mod)+65
        elif(op==2):
            c=(((ord(i)-65+b)*a)%mod)+97
        escritura(c,op)
    arch.close()

def escritura(c,op):
    if(op==1):
        archi=open("c.txt","a+")
    elif(op==2):
        archi=open("m.txt","a+")
    archi.write(chr(c))
    archi.close()

mod=int(input("Mod:  "))
alpha=int(input("alpha:  "))
beta=int(input("beta:  "))
op=int(input("1-encrypt 2- decrypt:  "))
if(op==1):
    lectura(op,mod,alpha,beta)
elif(op==2):
    res=euclidex(mod,alpha)
    print(res)
    if(res[1]<0):
        invalpha=mod+res[1]
    else:
        invalpha=res[1]
    print("alpha^-1="+str(invalpha))
    for i in range(0,mod):
        if(((beta+i)%mod)==0):
            b=i
    print("-beta="+str(b))
    lectura(op,mod,invalpha,b)
```

For the interface, I used html and JavaScript, so I had to rewrite the python code and logic in JavaScript.

This is the code for the encrypt function in html:

```
<!DOCTYPE html>
<html>
<title>Affine Cipher (Encrypt)</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="w3.css">
<body  background="background.jpg">

<div class="w3-card-4 w3-display-middle" style="width:50%;">
  <header class="w3-container w3-black">
    <h1>Affine Cipher (Encrypt)</h1>
  </header>
  <br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="message" id="message">
  </div><br/><br/><br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="mod" id="modulo">
  </div><br/><br/><br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="α" id="alpha">
  </div><br/><br/><br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="β" id="beta">
  </div><br/><br/><br/>
  <div>
    <button class="w3-qarter w3-button w3-black w3-padding-small" onclick="encrypt();">Encrypt</button>
  </div>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="ciphertext" id="ciphertext">
  </div><br/><br/><br/>
  <footer class="w3-container w3-black">
    <h5> . </h5>
  </footer>
</div>

<script>
function encrypt(){
  var modulo=parseInt(document.getElementById("modulo").value);
  var alpha=parseInt(document.getElementById("alpha").value);
  var beta=parseInt(document.getElementById("beta").value);
  res=euclidex(alpha,modulo);
  if(res[2]>1){
    alert("Invalid alpha, must be...gdc(alpha,modulo)=1");}
  else{
    var message=document.getElementById("message").value;
    var ciphertext="";
    for(var i=0;i<message.length;i++){
```

```
      ciphertextciphertext=ciphertext.concat(String.fromCharCode(((((message.charCodeAt(i)-97)*alpha)+beta)%modulo)+65));}
    document.getElementById("ciphertext").value=ciphertext;}
}

function euclidex(a,b){
  if(a%b==0){
    return [0,1,b];
  }
  else{
    var temp=euclidex(b,a%b);
    m=temp[0];
    n=temp[1];
    r=temp[2];
    return [n,m-parseInt(a/b)*n,r];
  }
}

</script>
</body>
</html>
```

And this is the code for the decrypt function

```
<!DOCTYPE html>
<html>
<title>Affine Cipher (Decrypt)</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="w3.css">
<body  background="background.jpg">

<div class="w3-card-4 w3-display-middle" style="width:50%;">
  <header class="w3-container w3-black">
    <h1>Affine Cipher (Decrypt)</h1>
  </header>
  <br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="ciphertext" id="ciphertext">
  </div><br/><br/><br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="mod" id="modulo">
  </div><br/><br/><br/>
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="α" id="alpha">
  </div><br/><br/><br/><!--Alfa debe ser primo relativo del modulo!!!-->
  <div>
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="β" id="beta">
  </div><br/><br/><br/>
  <div>
    <button class="w3-qarter w3-button w3-black w3-padding-small" onclick="decrypt();">Decrypt</button>
  </div>
  <div>
```

```
    <input class="w3-quarter w3-input w3-border" type="text" placeholder="plaintext" id="plaintext">
  </div><br/><br/><br/>
  <footer class="w3-container w3-black">
    <h5> . </h5>
  </footer>
</div>


<script>
function decrypt(){
  var modulo=parseInt(document.getElementById("modulo").value);
  var alpha=parseInt(document.getElementById("alpha").value);
  var beta=parseInt(document.getElementById("beta").value);
  res=euclidex(alpha,modulo);
  if(res[2]>1){
    alert("Invalid alpha, must be...gdc(alpha,modulo)=1");}
  else{
    if(res[0]<0){
        alpha=modulo+res[0];}
    else{
        alpha=res[0];}
    beta=modulo-(beta%modulo);
    var ciphertext=document.getElementById("ciphertext").value;
    var plaintext="";
    for(var i=0;i<ciphertext.length;i++){
      plaintextplaintext=plaintext.concat(String.fromCharCode(((((ciphertext.charCodeAt(i)-65+beta)*alpha))%modulo)+97));}
    document.getElementById("plaintext").value=plaintext;}
}

function euclidex(a,b){
  if(a%b==0){
    return [0,1,b];
  }
  else{
    var temp=euclidex(b,a%b);
    m=temp[0];
    n=temp[1];
    r=temp[2];
    return [n,m-parseInt(a/b)*n,r];
  }
}


</script>
</body>
</html>
```