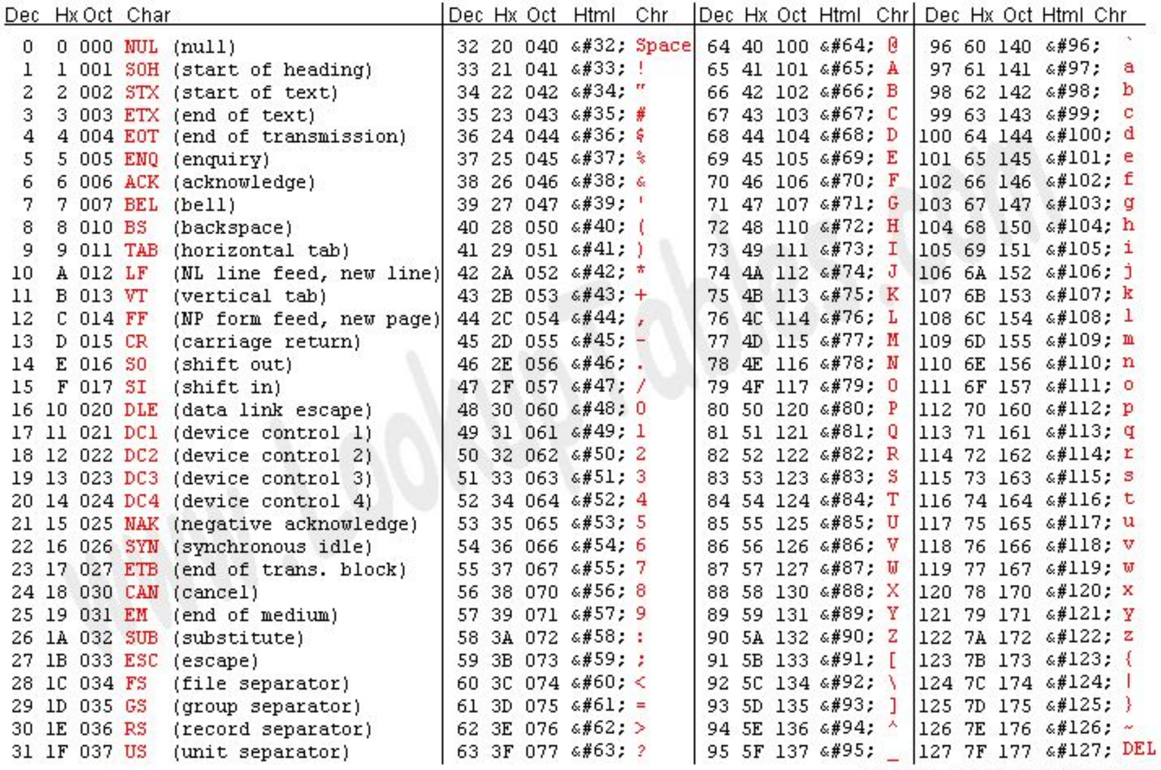
HY-457

Tutorial 1

Assignment 1

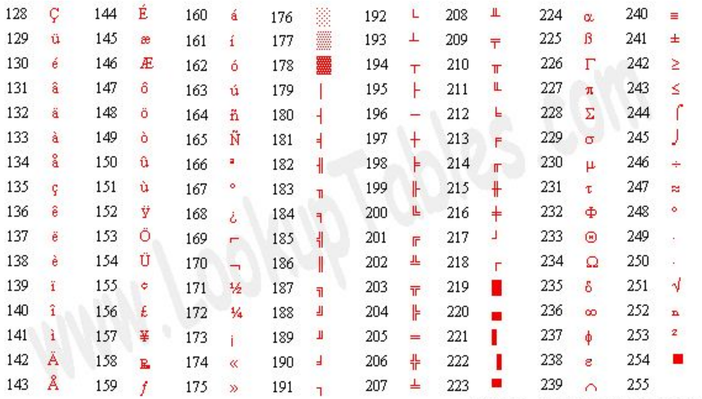
- In this assignment you will have to implement five simple ciphers
- You have to use C and implement them from scratch
- The main purpose of this assignment is to get you familiar with the internals and the process of implementing a cipher
- Also, this assignment will help you get familiar with some implementation tricks and development choices that appear when developing algorithms that seem trivial at first

The ASCII character set



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

Source: www.LookupTables.com



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

One-Time Pad (OTP)

- One-Time Pad (OTP) is a theoretically unbreakable cipher
- It requires a pre-shared key of at least the same length as the message
- The algorithm XORs each byte of the message with its respective key byte

Message: HelloWorld

Rand key: randombyte

Output: (H⊕r)(e⊕a)(l⊕n)(l⊕d)(o⊕o)(W⊕m)(o⊕b)(r⊕y)(l⊕t)(d⊕e) =

0x 3A 04 02 08 00 3A 0D 0B 18 01

One-Time Pad (OTP)

- The key will be generated using /dev/urandom
- You should store the encryption key in order to decrypt the message
- uint8_t * otp_encrypt(uint8_t *plaintext, uint8_t *key);
- uint8_t * otp_decrypt(uint8_t *ciphertext, uint8_t *key);
- The messages can only contain digits (0-9) and printable letters (A-Za-z)
- Be careful with non printable ASCII characters!

Caesar’s cipher

- One of the simplest and most known encryption techniques
- Each character of the input is replaced by the character found N-positions down the alphabet
- In order to decrypt the message, one has to know N

Message: hello

N: 4

Output: lipps

Caesar’s cipher

- uint8_t * caesar_encrypt(uint8_t *plaintext, ushort N);
- uint8_t * caesar_decrypt(uint8_t *ciphertext, ushort N);
- The messages can only contain digits (0-9) and printable letters (A-Za-z)
- Be careful with non printable ASCII characters!
- Be careful when you reach the end of the character set!

Playfair cipher

- The Playfair cipher encrypts pairs of letters (digraphs)
- The key is represented as a 5x5 matrix
- The letters of the key are placed in the matrix, from left to right beginning from the first row.
  - The rest of the alphabet’s letters are inserted in the grid alphabetically
  - Each letter is placed once in the grid.

Example: “HELLO WORLD”

H	E	L	O	W
R	D	A	B	C
F	G	I	K	M
N	P	Q	S	T
U	V	X	Y	Z

Playfair cipher

- Plaintext is separated in groups of two letters.
- If the number of the letters in the plaintext is even, the last letter is grouped with an ‘X’ character.
- If the letters of the group are the same the second letter is replaced with ‘X’.

Example: “WILL ATTACK AT DAWN” will result in

“WI LX AT TA CK AT DA WN”

Playfair cipher

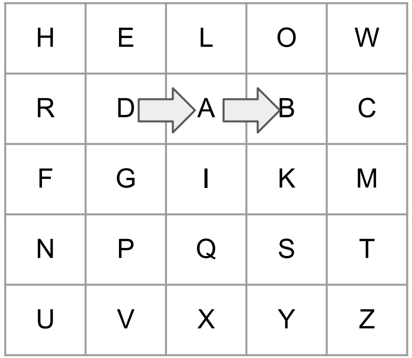
- If the letters appear on the same row of the key grid, they will be replaced with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).

- If the letters appear on the same column of the key grid, they will be replaced with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).

- If the letters are not on the same row or column, they will be replaced with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original

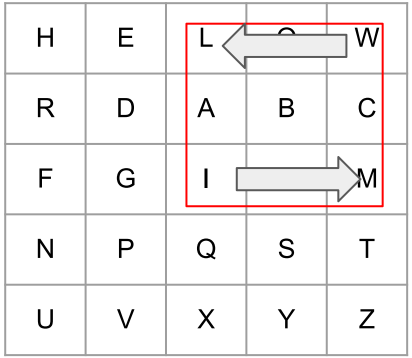
Playfair Cipher - examples

- **DA = AB**
  - **Same row**



H	E	L	O	W
R	D	A	B	C
F	G	I	K	M
N	P	Q	S	T
U	V	X	Y	Z

- **WI = LM**
  - **Rectangle**



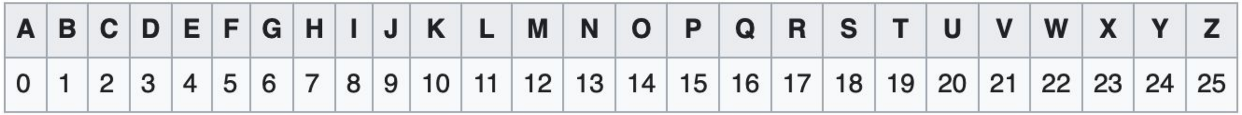
H	E	L	O	W
R	D	A	B	C
F	G	I	K	M
N	P	Q	S	T
U	V	X	Y	Z

Playfair Cipher

- unsigned char* playfair_encrypt(unsigned char *plaintext, unsigned char** key);
- unsigned char* playfair_decrypt(unsigned char *ciphertext, unsigned char** key);
- unsigned char** playfair_keymatrix(unsigned char *key);
- The messages can only contain capital letters (A-Z)
- Implement helper functions for preprocessing the plaintext

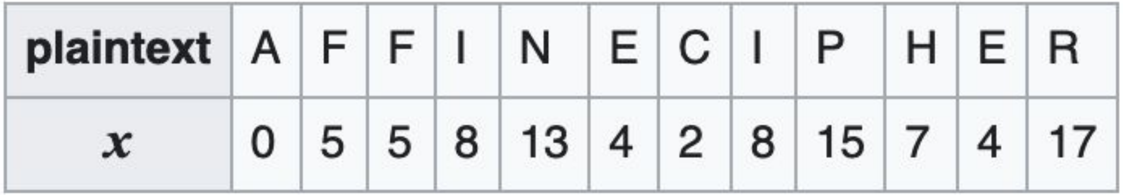
Affine cipher

- Each letter is mapped to its numeric equivalent.



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
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

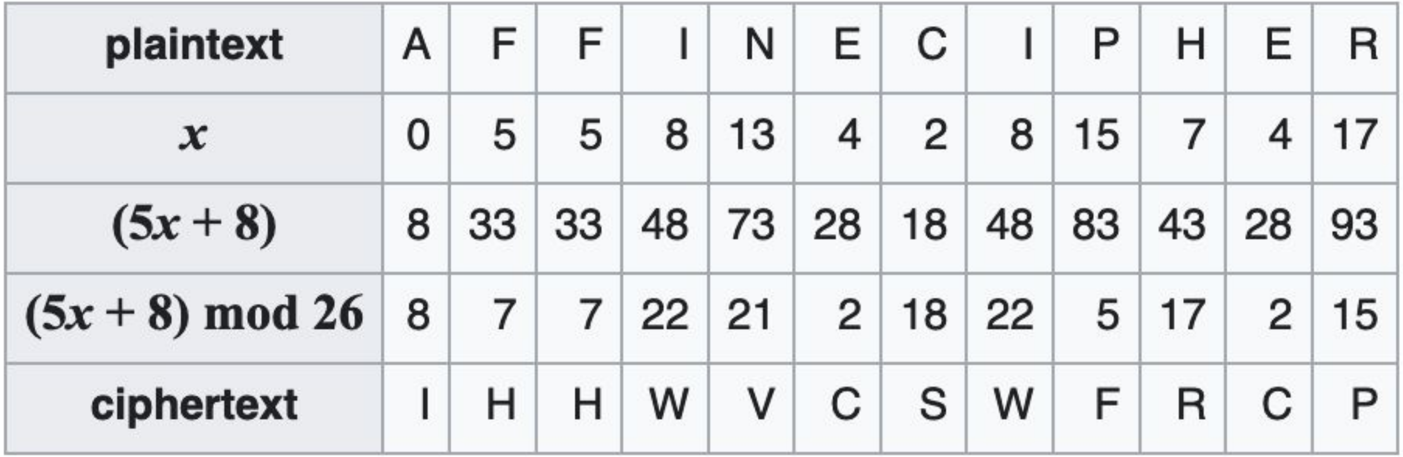
Message



plaintext	A	F	F	I	N	E	C	I	P	H	E	R
x	0	5	5	8	13	4	2	8	15	7	4	17

Affine cipher

Output



plaintext	A	F	F	I	N	E	C	I	P	H	E	R
x	0	5	5	8	13	4	2	8	15	7	4	17
$(5x + 8)$	8	33	33	48	73	28	18	48	83	43	28	93
$(5x + 8) \bmod 26$	8	7	7	22	21	2	18	22	5	17	2	15
ciphertext	I	H	H	W	V	C	S	W	F	R	C	P

- The character set contains only uppercase ASCII characters (A-Z)

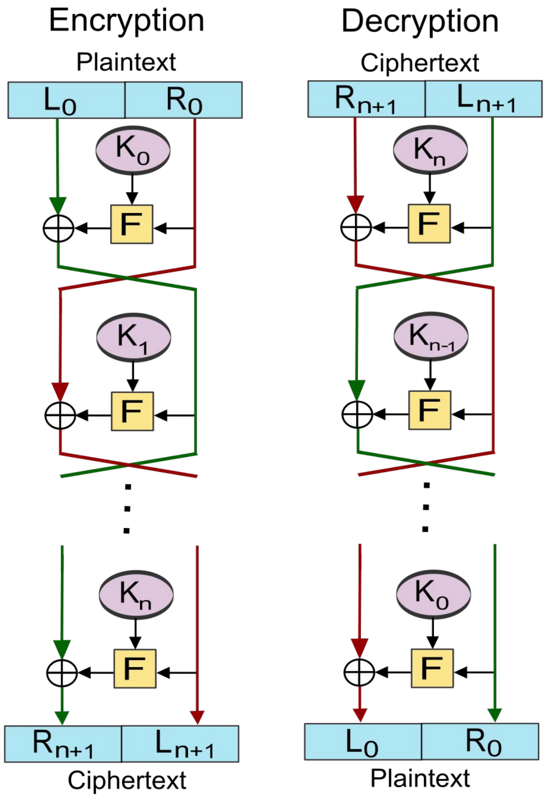
Affine cipher

- uint8_t * affine_encrypt(uint8_t *plaintext);
- uint8_t * affine_decrypt(uint8_t *ciphertext);

Feistel cipher

- A feistel cipher is a symmetric structure used in the construction of block ciphers.
- encryption and decryption are very similar operations, and both consist of iteratively running a function called a "round function" a fixed number of times
- The *round function* is a function which takes two inputs, a data block and a subkey, and returns one output the same size as the data block.

Feistel cipher



Feistel cipher

- uint8_t* round(uint8_t* block, uint8_t* key);
- uint8_t* feistel_encrypt(uint8_t* plaintext, uint8_t keys[]);
- uint8_t* feistel_decrypt(uint8_t* ciphertext, uint8_t keys[]);