HY457

Tutorial on Assignment 1

Konstantina Papafragkaki csdp1339

Assignment 1

- In this assignment you are asked to implement 6 ciphers and a simple Key-Value store.
- You have to use C to implement them from scratch.
- For part A (ciphers), you should create 2 files and a test file (demo):
  - cs457_crypto.h, containing function declarations and
  - cs457_crypto.c, containing the implementation of the functions
- For part B (Key-Value Store), you should create the file kv_store.c containing your implementation.

1. One-time pad

- It is a cryptographic cipher.
- It uses a predetermined random shared key that is at least as the size of the plaintext.
- The algorithm XORs each byte of the plaintext with the corresponding key byte.

1. One-time pad

- Use /dev/urandom (Linux based system) to generate a random key.
- Encryption is done by XORing the plaintext with the key and decryption by XORing the ciphertext with the key.
- Store the random generated key to use it for the decryption process.
- Assume that plaintext consists only of letters or numbers.

1. One-time pad encryption

Plaintext:	ThisIsACat
Key:	randombyte
Output:	(T⊕r)(h⊕a)(i⊕n)(s⊕d)(I⊕o)(s⊕m)(A⊕b)(C⊕y)(a⊕t)(t⊕e) =
Hex:	26 09 07 17 26 1E 23 3A 15 11

The decryption process reverses the encryption using the same key.

1. One-time pad implementation

- Implement the functions:
  - one_time_pad_encr and
  - one_time_pad_decr

These functions take as arguments the plaintext or ciphertext, its size, and the random generated key, and return the result of the operation.

2. Affine cipher

- ❖ It is a cryptographic cipher that uses mathematical functions for encryption and decryption to map letters to their equivalent counterparts.
- ❖ Encryption: (5x + 8) mod 26
- ❖ Decryption: 21(y - 8) mod 26
- ❖ Assume that the plaintext consists only of letters and/or spaces, and the program should handle letters in both upper and lower cases.

2. Affine encryption

Map each letter of the alphabet to its corresponding numeric value.

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

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)mod26	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 decryption inverses the current process.

2. Affine implementation

- Implement the functions:
  - affine_encr and
  - affine_decr

The functions take as arguments the plaintext / ciphertext and return the result accordingly.

3. Substitution algorithm decryptor

- Write a decryptor for the simple substitution algorithm, that decrypts a ciphertext without knowing the key.
  - Usage of the frequencies of characters in the ciphertext and the English Dictionary (https://github.com/dwyl/english-words) to detect word patterns (small recurring words such as “in”, “the” etc.)
  - Each iteration:
    - Takes as input a mapping (cipher alphabet -> alphabet) and prints the current plaintext
    - Takes as input a partially decrypted word and prints the matching words
- The case (upper/lower) of each letter of the ciphertext remains the same in the plaintext.
  - To make the process easier, you can convert the ciphertext to uppercase/lowercase and restore the case in the generated plaintext.

3. Substitution algorithm decryptor

An example:

Ciphertext: Zrwu wu i uwqflc ctiqflc hap zrc zezapwil za urao zrc euimc ah zrwu ilmapwzrq.

Detect small words that may be common from the frequency of the English Dictionary and the ciphertext:

- “wu” has multiple occurrences
- can be the word “an”
- replace w with a and u with n
- repeat this process until the original message can be retrieved

3. Substitution algorithm decryptor

- a → w

  a* a* * a* ******* *** *** *****a** ** **** *** ***** ** **a* *****a***.

- n → u

  an an * na* ******* *** *** *****a** ** n*** *** *n*** ** **an *****a***.

4. Trithemius cipher

- It is a straightforward polyalphabetic substitution cipher.

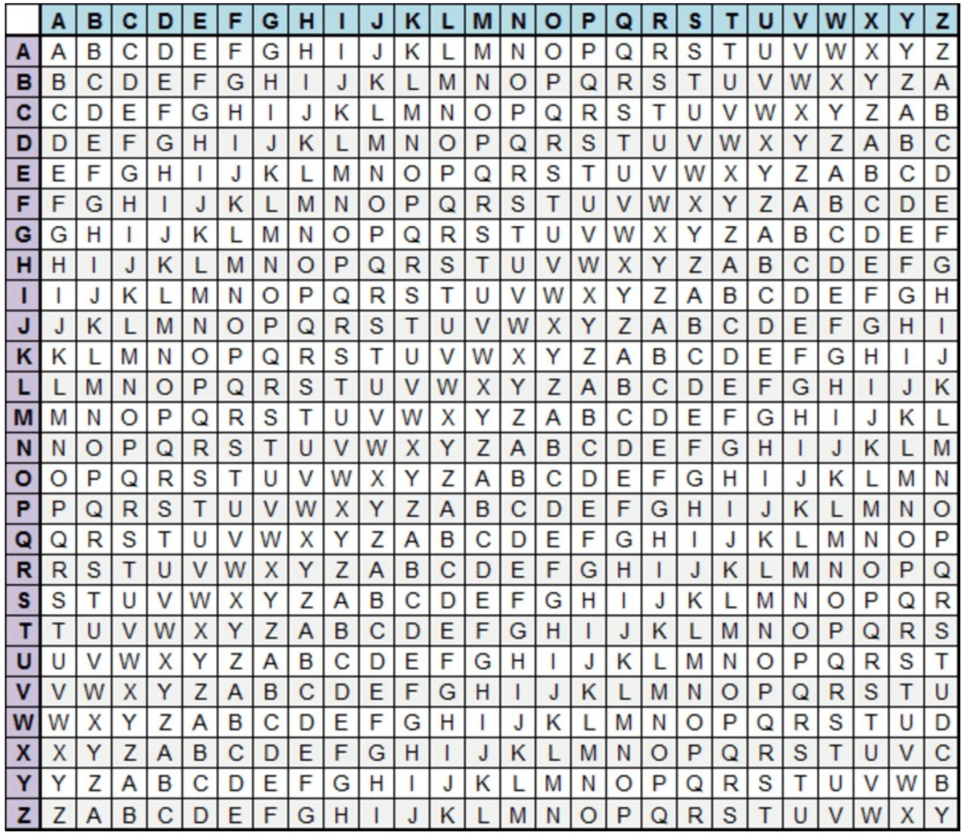
- It uses an alphabetic table, also known as tabula recta for both encryption and decryption processes.

- During encryption, each subsequent letter in the plaintext is shifted one position forward from the previous one. Conversely, during decryption, the algorithm reverses this process to recover the original plaintext.

4. Trithemius encryption

- Plaintext:
  - HELLO (after omitting spaces and punctuation, the plaintext can be in upper case)
- Ciphertext:
  - HFNOS
  - The letter corresponding to the letter ‘H’ is located at the **first row**, so it is ‘H’ as well.
  - For the letter ‘E’ in the **second** position of the plaintext, the associated letter is located in the **second** row and column of ‘E’ and it is ‘F’.
  - Continue until the ciphertext is generated.

The decryption inverses the current process and adds the spaces and punctuation.



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

4. Trithemius implementation

- Implement the functions:
  - trithemius_encr and
  - trithemius_decr

The functions take as arguments the plaintext / ciphertext and return the result of the operation.

5. Scytale cipher

- It is a transposition cipher that involves a cylinder with a strip of parchment wound around it, containing the written message.

- The recipient utilizes a rod of the same diameter on which the parchment is wrapped, to read the message.

- It is essential to store the number of rods in memory for both encryption and decryption processes.

5. Scytale encryption

- Suppose we have 5 rods (number of columns).
- Initial text: “I am hurt very badly help”
- Plaintext after omitting the spaces and punctuation:
  - “Iamhurtverybadlyhelp”
- Ciphertext after unwinding across the rows:
  - “Iryyatbhmvaehedlurlp”

The scytale:

```
      |   |   |   |   |   |   |
      | I | a | m | h | u |   |
    __| r | t | v | e | r |__|
   |  | y | b | a | d | l |
   |  | y | h | e | l | p |
   |  |   |   |   |   |   |
```

5. Scytale decryption

- Suppose we have 5 rods (number of columns).
- Ciphertext:
  - “Iryyatbhmvaehedlurlp”
- Every fourth letter will appear on the same line:
- Plaintext after re-insertion of spaces:
  - “I am hurt very badly help”

The scytale:

```
     |   |   |   |   |   |  |
     | I | a | m | h | u |  |
   __| r | t | v | e | r |__|
  |  | y | b | a | d | l |
  |  | y | h | e | l | p |
  |  |   |   |   |   |   |
```

5. Scytale implementation

- Implement the functions:
  - scytale_encr and
  - scytale_decr

The functions take as arguments the plaintext / ciphertext, the diameter of the rod and return the result of the operation.

6. Rail-Fence cipher

- It is a classical type of a transposition cipher.

- The number of rails is given to both encryption and decryption.

6. Rail-Fence encryption

- Initial text: “WE ARE DISCOVERED. RUN AT ONCE.”
- Suppose that we have 3 “rails”, then:
- Plaintext after removing the spaces and punctuation from the initial text:
  - “WEAREDISCOVEREDRUNATONCE”
- Ciphertext:
  - “WECRUO ERDSOEERNTNE AIVDAC” after reading off the text horizontally and added space when finished reading a rail.

The text is written like that:

```
W . . . E . . . C . . . R . . . U . . . O . . .
. E . R . D . S . O . E . E . R . N . T . N . E
. . A . . . I . . . V . . . D . . . A . . . C .
```

During decryption, we find the “rails” and then create the same array at the left to read it diagonally. Then we add the spaces and punctuation to generate the initial text.

6. Rail-Fence implementation

- Implement the functions:
  - rail_fence_encr and
  - rail_fence_decr

The functions take as arguments the plaintext / ciphertext, the number of rails for the encryption process and return the result of the operation.

7. Key-Value Store

- It is an application used to store and manage (key,value) pairs in a database.
- It securely stores each (key,value) pair in the database file.
- You have to implement it using C and the OpenSLL library.
- Support of 2 functionalities: **add**, retrieve info (**read** and **range-read**).

7. Key-Value Store

- $ kv add -f <filename> key value
  - Read from the command line the key and value and stores them in the database file.
  - If the file exist, the new entry should be appended at the end. If not, the file should be created.
  - The (key,value) pair should be saved in a CSV format.
  - Assume that both key and value are positive integers.

7. Key-Value Store

- $ kv add -f <filename> key value
    - Before saving the pair, we have to encrypt it using AES encryption.
    - The whole file should be encrypted too, again using AES.
    - The user should provide a master password that will be use to generate the key and IV for the encryptions.

7. Key-Value Store

An example:

```
$ kv add -f db.txt 1 3
$ Enter password: pass
```

After that, the file db.txt should look like this:

```
key,value
<encrypted key>,<encrypted value>
```

7. Key-Value Store

- $ kv read -f <filename> key
  - Searches the database file for the key. To see if the key exists, you must decrypt the keys in the database file.
  - If it exists, it prints the pair to the console.
  - The value should be decrypted too before displayed.

7. Key-Value Store

An example:

```
$ kv read -f db.txt 1
$ Enter password: pass
```

The program should print:

```
Key: 1 has value: 3
```

7. Key-Value Store

- $ kv range-read -f <filename> key1 key2
  - The key1 should be less or equal to key2.
  - Searches the database file for the range of keys. To see if each key exists, you must decrypt the keys in the database file.
  - If the current key exists, it prints the pair to the console.
  - Each value should be decrypted too before displayed.

7. Key-Value Store

An example:

```
$ kv range-read -f db.txt 1 3
$ Enter password: pass
```

The program should print:

```
Key: 1 has value: 3
```