KAUST
King Abdullah University of Science and Technology

Programming Languages Project

CS 242

Cryptography in D Language

Instructor: Prof. Mohammed ElRamly

Name: Omniah Hussain Nagoor
ID: 123923
Name: Shahzeb Siddiqui
ID: 124211

Introduction:

™ Cryptography is a technique for distorting the message to protect against others from reading it through the use of encryption. The goal of cryptography is to ensure confidentiality, data integrity, and authentication. Modern cryptography is based on mathematical theory and computational complexity.

In this project we implemented three different encryption algorithms. These algorithms are Caesar cipher, Stream cipher and Hill cipher. The purpose of implementing these different encryption algorithms is to demonstrate D special features.

Design and Structure of the project implementation:

I. Caesar Cipher

Caesar Cipher is a substitution cipher algorithm with a key which represents the shift amount in the table mapping which affects how letters are substitution during cipher. Our implementation of Caesar Cipher requires a plaintext, a key and outputs a ciphertext.

D has a powerful feature known as array-slicing which can reference subarray reference for array operations. In Caesar Cipher, Array Slicing was helpful in creating a table mapping with the key shift.

The variable **table** is a character array which represents letters in the plaintext that will be mapped with the variable **newtable**. The **newtable** is a dynamic array that references same array **table** but its reference are shifted by the key elements. The first segment table[key .. charCount] references subarray from index key to charCount which is the end of the array. The binary operator **~** concatenates two arrays. In order to make table circular, we need to concatenate with the rest of the elements (**table [0 .. key]**). After the mapping is created, we cipher the plaintext and check if character is in **table** and map it to the character in **newtable.** The ciphertext is created dynamically using the **~** operator.

```
void ceasar_cipher(int key)
{


  write("Enter Plaintext: ");
  string plaintext = stdin.readln();

  int i,j;
  char table[charCount] = "abcdefghijklmnopqrstuvxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*(),./;'[]{}:";
  char[] newtable = table[key .. charCount] ~ table[0 .. key];  // Array Slicing - means specify the subarray and create new pointer refrence to it
                                                                // Decompose the array from key to the end and append it with the rest
  char[] plaintext_inchar = plaintext.dup;                      // Doulbecate the content of the array into a dynamic array
  char[] ciphertext;


  for (i = 0; i < plaintext_inchar.length; i++)
  {
    for (j = 0; j < table.length; j++)
    {
        if (plaintext_inchar[i] == table[j])
        {
            ciphertext ~= newtable[j];
        }
    }
  }

  writeln("key = ",key);
  writeln("table    = ",table);
  writeln("newtable = ",newtable);
  writeln("plaintext =  ", plaintext);
  writeln("ciphertext = ",ciphertext);

}
```

Caesar cipher is a simple cipher algorithm which can be easily breakable using frequency analysis of letters. We decided to make Caesar cipher stronger, by using multiple keys and exploiting additional features from D such as variadic and nested function. The variadic function for caesar cipher is defined to take a plaintext with a dynamic array of keys. The function definition is the following:

void variable_keys_ceaser_cipher(string plaintext,in int[] keysarray ...)

The same caesar cipher algorithm was nested inside this function since the implementation is exactly the same but the function parameters are different. The function definition of caesar cipher is the following:

void ceasercipher(char letter, int key, int keyindex)

This function will now operate on a letter and a unique key such that the plaintext is ciphered with a different key which eliminates the fact that same letter can be mapped to the same cipher letter. The algorithm iterates over plaintext characters and uses a different key and applies the caesar cipher algorithm. The cipher text is created dynamically like the previous implementation.

```
/*******************************************************
 *  variable_keys_ceaser_cipher function:              *
 * This function take a plain text and cipher it with  *
 * multiple number of keys which ciphers each character*
 * in the plaintext by a different key in the table    *
 *******************************************************/
void variable_keys_ceaser_cipher(string plaintext, int[] numkeys ...) // varaidic function parameters
{
    int i, index;

    char table[charCount] = "abcdefghijklmnopqrstuvxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*(),./;'[]{}:";

    char [][charCount] newtable;
    char [] plaintext_inchar = plaintext.dup;             // Dublicate the content of the array into a dynamic array
    char [] ciphertext;


     /*******************************************************
      *  ceasarcipher function:                             *
      * This function take a character and cipher it with   *
      * the given key which represtent the shift amount it  *
      * the table                                           *
      *******************************************************/
    void ceasercipher(char letter, int key, int keyindex) // Nested Function Implementation
    {
        int j;

        for (j = 0; j < table.length; j++)
        {
            if (letter == table[j])
                ciphertext ~= newtable[keyindex][j];
        }

        writeln("letter = ",letter,"\tciphertext = ",ciphertext, "\t key = ", key);
    }
```

```
    writeln("table      = ",table);
    for (i = 0; i < numkeys.length; i++)
    {
      newtable[i] = table[numkeys[i] .. charCount] ~ table[0 .. numkeys[i]];
      writeln("newtable[",i,"]= ",newtable[i], "\tkey = ",numkeys[i]);
    }

    index = 0;

    writeln("\nEncrypting Plain text = ", plaintext);


    while(index < plaintext_inchar.length)                          // Encrypt the plain text char by char
    {
        for (i = 0; i < numkeys.length; i++)
        {
            if (index == plaintext_inchar.length)
                break;
            else
            ceasercipher(plaintext_inchar[index++],numkeys[i],i); // Enc each char with the given key
        }
     }

}
```

II. Hill Cipher

Hill cipher is a polygraphic substitution cipher invented by Lester Hill in 1929. It is an example of block cipher in which it encrypts a block of letters simultaneously. Hill cipher utilizes concepts of linear algebra. Our implementation of Hill Cipher requires a plaintext, a key matrix and outputs a ciphertext.

In Hill cipher letters are joined in an array which known as a table. In this table each letter is mapped to a number. **Associative array** is one of the powerful features of D language that help in this implementation. The **Associative array** is an array which has an index that is not necessary to be of integer type. This index is known as a key and it maps to a value of the define type. In our implementation each letter is a key which maps to it numerical value using the **Associative array** feature. Another feature of D language that makes the implementation easy and more flexible is the **dynamic array**. The **dynamic array** is an array with variable number of elements that can be assigned at run time. This feature is helpful in declaring the array that takes the user plaintext.

In the beginning the key matrix will randomly initialized using the **uniform()** function. This function returns a random number between the two parameters.

```
for(int i; i < KeySize; i++)
{
  key ~= uniform(0,s.length);
}
```

In order to encrypt the user plaintext it length has to be checked first. The plaintext length has to be divisable by the key matrix width. Since the plaintext will be subdivided into subparts (vectors). Each vector will be multiplied by the key matrix. The final result need to be modulo with the size of the table elements. Finally, each numerical value will be mapped again to its corresponding letter.

The following figure shows an example of vector matrix multiplication:

$$C1 = 9*p1 + 18*p2 + 10*p3 \pmod{26}$$
$$C2 = 16*p1 + 21*p2 + 1*p3 \pmod{26}$$
$$C3 = 5*p1 + 12*p2 + 23*p3 \pmod{26}$$

$$\begin{pmatrix} C1 \\ C2 \\ C3 \end{pmatrix} = \begin{pmatrix} 9 & 18 & 10 \\ 16 & 21 & 1 \\ 5 & 12 & 23 \end{pmatrix} \begin{pmatrix} p1 \\ p2 \\ p3 \end{pmatrix} \pmod{26}$$

III. Stream Cipher

Stream Cipher is a symmetric key cipher where the plaintext is XOR with a pseudo random key. The key should be at least the same size or greater than the plaintext. The larger the key size, the harder it is to break the encryption.

Our implementation used a 64 bit stream cipher because the data type unsigned long (ulong) is a 64 bit representation in machine code. Our function streamcipher took a key which represents a private key. Each private key is associated with a person and no two people have the same key. This function was called by different private key each time with the same packet 10E18 and the encrypted packet was different at every call. The binary operator **^** is the xor operation.

```
void streamcipher(ulong key)
{
    ulong Packet = 1000000000000000000;
    ulong EncryptedPacket;

    EncryptedPacket = Packet ^ key;
```

IV. Generate Private Keys

In symmetric key encryption, both sender and receiver share the same private key which is used in stream cipher. In order to use stream cipher, we decided to create functionality for generating a random key and associating it to a different person. We utilized many features in this implementation such as Associative Arrays, nested function, and contract programming.

The function **gen_key** takes an argument keyname which represents the name of the person who owns the key. The return type of this function is ulong since it will generate a private key 64 bits in length. Initially, the function generates a random key using the D function **uniform(min,max)** which creates a random number between min and max value. Afterwards, we call **check_if_key_valid(key)** to ensure uniqueness of key. The function checks the pre-conditioner which is the key value to make sure it's in the range of (0,long_max) to guarantee key length is 64 bits. The code block delimited by **in** denotes the pre-conditioner statement, and the function **assert(expr)** evaluates the expr to either true or throws an exception of type AssertError. If the expr is true, the function will proceed with the body code block which is implemented after the pre-conditioner is satisfied. This function checks if key is in keylog and determines if a new key needs to be generated if there is a keymatch. The function's return type is ulong which can return the same key or new key depending if a keymatch occurs. The array **keylog** is an associative array which means, the array index is not by **int** which is the typical convention. The index in keylog is based on **string** since we decided to index keylog by person's name.

```
/*****************************************************
*  gen_key function:                                 *
*   This function generates a private key for each   *
*   person using contracts and stores the key in a   *
*   keylog to ensure unique key everytime            *
*****************************************************/
ulong gen_key(string keyname)
{
  ulong check_key_if_valid (ulong key)                // Another example of Nested Functions
                                                      // Assert - Check if the key between 0 and  long_max
    in { assert (key >= 0 && key <= long_max);}       // contract programming: preconditioner check for key value between 0 - long_max
    body                                              // contract programming: execute body which generates a unique key everytime
      {
        bool keymatch = false;                        // check if the new key exists in the keylog

        if (keylog.length == 0)         // The first key is added to keylog without any checks
            keylog[keyname] = key;
        else                            // Check if private key is unique (not in keylog)
        {
            do
            {
                for (int i = 0; i < numkeys_assign - 1; i++)        // looping over keys in keylog to check if key is unique
                {
                    if (keylog[person_with_key[i]] == key)
                        keymatch = true;
                }

                if (keymatch == true)                               // if there is keymatch in keylog, create new key
                {
                    key = uniform (0,long_max);                     // create new random key for uniqueness
                }

            }
            while (keymatch == true);                               // loop if there is a matched key in keylog
            keylog[keyname] = key;
        }
        return key;
      }
```

```
  ulong key = uniform (0, long_max);                                // create initial key
  ulong userkey = check_key_if_valid(key);                          // generate unique private key

  return userkey;

}
```

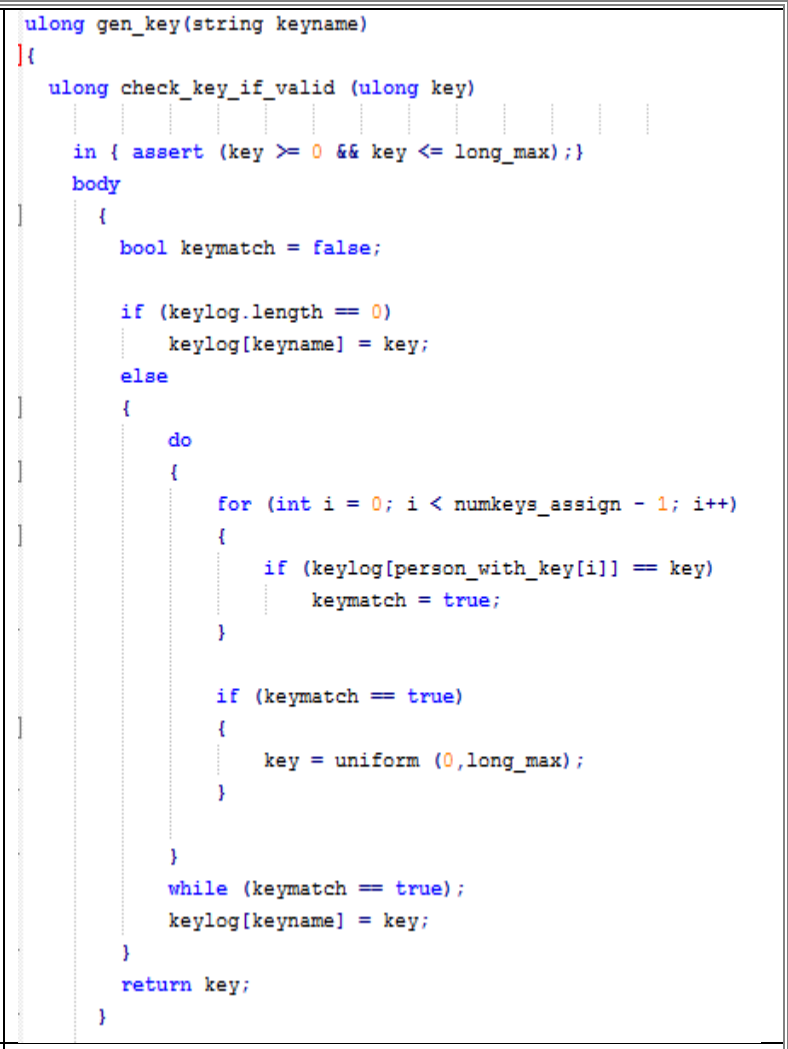
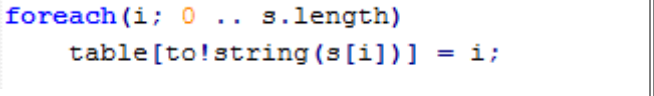
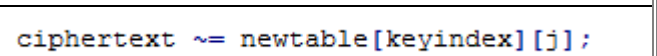
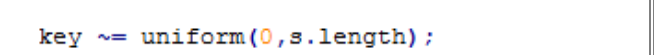
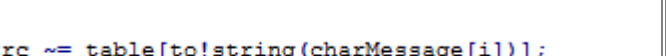
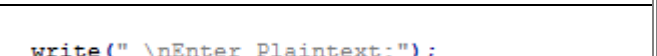
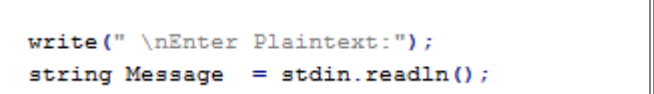
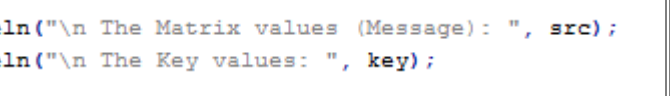
Implemented features:

Feature	Description	Example
Associative Array	Associative array is an array which has an index that is not necessarily to be an integer. It can be other type such as string.	`int [string] table;`

Dynamic Array	Dynamic Array has variable number of elements which assigned at run time.	`int [] key;`
		`int [] src;`
		`int [] sum;`

Variadic Function	Variadic Function is a function that takes variable number of arguments.	`void variable_keys_ceaser_cipher(string plaintext, int[] numkeys ...)`
Array Slicing	Array slicing specifies subarray and create new pointer reference to it.	`char[] newtable = table[key .. charCount] ~ table[0 .. key];`
.length	.length is array operation that returns the number of values in the array.	`sum[i] = sum[i] % s.length;`
.dup	.dup is array operation that creates a dynamic array of the same size and copies the contents of the array into it.	`char EncMssg[] = charMessage.dup;`
.reverse	.reverse is array operation that reverses in place the order of the elements in the array and returns the array.	`binaryreverse = binary.reverse;`

Nested Function	Nested Function is function inside another function	`void variable_keys_ceaser_cipher(string plaintext, int[] numkeys ...`
		`{`
		`int i, index;`
		`char table[charCount] = "abcdefghijklmnopqrstuvxyz ABCDEFGHIJKLMN`
		`char [][charCount] newtable;`
		`char [] plaintext_inchar = plaintext.dup;`
		`char [] ciphertext;`
		`void ceasercipher(char letter, int key, int keyindex)`
		`{`
		`int j;`
		`for (j = 0; j < table.length; j++)`
		`{`
		`if (letter == table[j])`
		`ciphertext ~= newtable[keyindex][j];`
		`}`
		`writeln("letter = ",letter,"\tciphertext = ",ciphertext, "\t`
		`}`

Contract Programming	**Assert(expr)** evaluates expression expr if it is nonzero there no effect otherwise assert throws an exception of type AssertError. **Contract Programming** preconditioner check for the Expression and if it is true then execute the body.	
Foreach()	Foreach is an iterator that start by index 0 to the end of the array	
~	**~** This binary operator is used to concatenate array elements	
Uniform (min,max)	uniform(min,max) give a random number between [min,max]	
to!string()	**to!string()** converts a value from type Source to type target.	
readln()	**readln()** read line from stream fp. It return a string data type.	
write()	**write()** prints the given arg to the standard output. A call without any arguments will fail to compile.	
Writeln()	**writeln()** Similar to write function but with a newline. Calling writeln without arguments is valid and just prints a newline to the standard output.	

Comparison between D features with other languages:

	C	C++	D	Java
OO	No	Yes	Yes	Yes
Multiple Inheritance	No	Yes	No	No
Operator Overloading	No	Yes	Yes	No
Garbage Collection	No	No	Yes	Yes
Standard Documentation Mechanism	No	No	Yes	Yes
Function Pointers	Yes	Yes	Yes	No
Interfaces	No	Yes	Yes	Yes
Contract Programming	No	Yes	Yes	Yes
Nested Function	No	No	Yes	No

References:

- *D programming language*. (n.d.). Retrieved from http://dlang.org/
- Andrei Alexandrescu, “The D Programming Language”.
- *Shift (caesar) cipher*. (n.d.). Retrieved from http://cryptoclub.math.uic.edu/shiftcipher/shiftcipher.htm