ema 9. Criptografie

(1.) Implementați criptosistemul Rabin.
(2.) Implementați criptosistemul Massey-Omura.
(3.) Implementați criptosistemul Merkle-Hellman
(4.) Implementați problema rucsacului în general.

15. Alice utilizează un criptosistem Merkle-Hellman pe un alfabet cu 26 de caractere (literele A-Z), unitățile de mesaj având un caracter. Cheia publică a lui Alice este șirul {34, 51, 58, 11, 39} iar cheia secretă este (b = 18, m = 61). Criptați mesajul WHY și apoi decriptați-l

---

```
1.  #include <iostream>
    #include <string>
    #include <unordered_map>
    using namespace std;
  // funcția pentru a crea o hartă de substituție bazată pe o cheie
    unordered_map <char, char> createSubstitutionMap(const string& key){
            unordered_map <char, char> substitutionMap;
            char substitutionChar = 'A';
            for (char ch : key){
                    if (substitutionMap.find(ch) == substitutionMap.end()){
                            substitutionMap [ch] = substitutionChar++;
                    }
            } return substitutionMap;
    }
  // funcția pentru criptarea unui text folosind o hartă de substituție
    string encrypt(const string& text, const unordered_map<char, char>&
substitutionMap){
            string encryptedText;
            for (char ch : text){
                    if(substitutionMap.find(ch) != substitutionMap.end()){
                        encryptedText += substitutionMap.at(ch);
                    }else{
```

```
            encryptedText += ch;
        }
    }return encryptedText;
}
// funcția pentru a inversa harta de substituție
unordered_map<char, char> invertSubstitutionMap(const unordered_map<char,
char>& substitutionMap){
        unordered_map<char, char> inverseMap;
        for(auto pair : substitutionMap){
            inverseMap[pair.second]= pair.first;
        }
        return inverseMap;
}
// funcția pentru decriptarea unui text folosind o hartă de substituție inversată
string decrypt(const string& encryptedText, const unordered_map<char, char>&
inverseMap){
        string decryptedText;
        for(char ch: encryptedText){
            if(inverseMap.find(ch) != inverseMap.end()){
                decryptedText += inverseMap.at(ch);
            }else{
                decryptedText += ch;
            }
        }return decryptedText;
}
int main(){
    // Exemplu
      string text= "Hello World"; string key= "Key";
    // crearea hărții de substituție
      unordered_map<char, char> substitutionMap = createSubstitutionMap(key);
    // criptarea textului:
      string encryptedText = encrypt(text, substitutionMap);
      cout<< "Text criptat: "<< encryptedText <<endl;
    // crearea hărții de substituție inversată
       unordered_map <char, char> inverseMap = invertSubstitutionMap
(substitutionMap);
    // decriptarea textului
       string decryptedText = decrypt (encryptedText, inverseMap);
       cout << " Text decriptat:  " << decryptedText << endl;
       return 0;
 }

2.  #include <iostream>
    #include <cmath>
    #include <cstdint>
    using namespace std;
    // funcția pentru a calcula (base^exp) % mod folosind exponențiere
modulară rapidă
     uint64_t modExp (uint64_t base, uint64_t exp, uint64_t mod){
          uint64_t result = 1;
          while (~~exp~~ exp > 0){
              if (exp % 2 == 1){
                  result = (result * base) % mod;
              }
              base = (base * base) % mod;
              exp /= 2;
          } return result;
     }
    // funcția pentru a calcula inversul modular folosind algoritmul extins al lui
Euclid
     int64_t modInverse (int64_t a, int64_t m){
          int64_t m0 = m, t, q;
          int64_t x0 = 0, x1 = 1;
          if (m == 1) return 0;
          while (a > 1) {
               q = a/m;
               t = m;
               m = a % m;
               a = t;
               t = x0;
               x0 = x1 - q * x0;
               x1 = t;
          }
```
Scanat cu CamScanner

```
    if(x1<0)
        x1 += m0;
    return x1;
}
int main(){
    //Exemplu
    // Alegem 2 numere prime mari
    ~~unit~~ uint64_t p= 47;
         uint64_t q = 53;
         uint64_t n=p*q; //n este modulul comun
    // Alegem cheile publice și private
      uint64_t e1= 17; // cheia publică a lui Alice
      uint64_t e2 = 19; // cheia publică a lui Bob
      uint64_t phi_n = (p-1)*(q-1);
      uint64_t d1 = modInverse (e1, phi_n); //cheia privată a lui Alice
      uint64_t d2= modInverse (e2, phi_n); //cheia privată a lui Bob
    // Mesajul de criptat
      uint64_t M=1234; // mesajul original
    // pas 1: Alice criptează mesajul folosind e1
      unit64_t C1 = modExp (M, e1, n);
      cout<<"Mesajul criptat de Alice:  "<<C1<<endl;
    // pas 2: Bob criptează mesajul criptat de Alice folosind e2;
      unit 64_t C2 = modExp ( C1, e2, n);
      cout<<"Mesajul dublu criptat de Bob:  "<<C2<<endl;
    // pas 3: Bob decriptează mesajul folosind d2;
      ~~cout~~ uint64_t C3 = mod Exp (C2, d2, n);
       cout<<"Mesajul decriptat de Bob:  "<< ~~end~~ C3<endl;
    // pas 4: Alice decriptează mesajul folosind d1:
       uint64_t ~~C4~~ = decryptedMessage = modExp(C3, d1, n);
       cout<<"Mesajul decriptat de Alice :  "<< decryptedMessage<<
    endl;
    return0;
}
```

```
3. # include < iostream>
   # include < vector>
   # include < numeric >
   # include < algorithm>
   using namespace std;

 // funcția pentru generarea unei secvențe supercrescătoare
 vector <int> generateSuperIncreasingSequence (int n) {
       vector<int> sequence (n);
       sequence [0] = rand () % 10 + 1; // valoarea inițială între 1 și 10
       for (int i = 1; i<n; i++) {
             sequence [i] = accumulate (sequence . begin (), sequence . begin () + i, 0)
 + (rand () % 10 + 1);
       } return sequence;
 }
 // funcția pentru găsirea unei W1 coprim cu M
 int findCoprime (int M) {
       int W1 = rand () % (M - 1) + 1;
       while (_gcd (W1, M) != 1) {
             W1 = rand () % (M-1) + 1;
       } return W1;
 }
 // funcția pentru calcularea inversului modulo
 int modInverse (int a, int M) {
       for ( int x = 1; x <M; x++) {
             if ((a*x) % M == 0) {
                   return x;
             }
       } return 1;
 }
 // funcția pentru generarea cheii publice
   vector <int> generatePublicKey (const vector<int> & privateKey, int M, int W1) {
         vector <int> publicKey (privateKey. size ());
         for (size_t i = 0; i< privateKey. size (); i++) {
               publicKey [i] = (privateKey [i] * W1) % M;
         } return publicKey;
   }
```

```
//funcția pentru criptarea mesajului
 int encrypt (const vector<int> & publicKey, const string & message){
      int encryptedMessage = 0;
      for( size_t i=0; i< message.size(); i++){
            if(message [i] == '1'){
                  encryptedMessage+= publickey [i];
            }
      } return encryptedMessage;
 }
 //funcția pentru decriptarea mesajului
 string decrypt (const vector<int> & privateKey, int encryptedMessage,
int M, int W){
         int W_inv = modInverse (W, M);
         int cPrime = (encryptedMessage * W_inv) % M;
         string decryptedMessage (privateKey.size(), '0');
         for( int i= privateKey.size()-1; i>=0; i--){
               if( privateKey[i] <= cPrime){
                     cPrime -= privateKey [i];
                     decryptedMessage [i] = '1';
               }
         } return decryptedMessage;
 }
 int main(){
        srand (time(0));
        //Exemplu
        // generarea cheii private
        int n = 8; // lungimea cheii
        vector<int> privateKey = generateSuperIncreasingSequence(n);
        int M = accumulate (privateKey.begin(), privateKey.end(), 0) + rand()%
10 +1;  int W = findCoprime(M);
        //generarea cheii publice
        vector<int> publickey = generatePublicKey (privateKey, M, W),
        // mesajul de criptat
          string message= "10101010";
        // criptarea mesajului
int encryptedMessage = encrypt(publicKey, message);
cout << "Mesaj criptat: " << encryptedMessage << endl;
//decriptarea mesajului;
string decryptedMessage = decrypt(privateKey, encryptedMessage, M, |X|);
cout << "Mesaj decriptat: " << decryptedMessage << endl;
return 0;
}
```

4)
```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
bool isSupercrescător (vector<int> & values){
        int sum = 0;
        for(int i = 0; i<values.size(); i++){
                if (values[i] <= sum){
                        return false;
                } sum += values[i];
        }return true;
}
//funcția auxiliară pt. backtracking
void backtrack (vector<int> & values, vector<int> & selected, int totalValue,
int index, int capacity, vector<int> & bestSelection){
        if (index >= values.size() || totalValue > capacity){
                return;
        }
        //verificăm dacă adăugarea valorii curente formează un șir
supercrescător
        if(isSupercrescător(selected) && totalValue > accumulate(
bestSelection.begin(), bestSelection.end(), 0)) {
                bestSelection = selected;
        }
        // încercăm să adăugăm valoarea curentă
        selected.push_back(values[index]);
        totalValue += values[index];
```

Scanat cu CamScanner

```
        backtrack (values, selected, totalValue, index+1, capacity, bestSelection);
        //renunțăm la valoarea și curentă și încercăm cu următoarea
        selected .pop. back ();
        total Value  -= values [index];
        backtrack (values, selected, totalValues, index+1, capacity, bestSelection);
    }
    void backpack (vector <int> & values, int capacity){
        sort (values ~~<int> & values~~, values. end());
                  .begin ()
        vector <int> selected, bestSelection;
        backtrack (values, selected, 0, 0, capacity, bestSelection);
        if (! bestSelection. empty ()) {
                cout<< " Valoarea max in rucsac: "<< accumulate (
bestSelection. begin(), bestSelection. end (), 0) << endl;
                cout<< " Obiectele din rucsac: ";
                for (int i=0; i< bestSelection . size(); i++){
                    cout<< bestSelection [i] << " ";
                } cout<< endl;

        } else {
                cout<< " Nu există o selecție care să formeze un
șir supercrescător " << endl;
        }
    }
    int main (){
        vector<int> values = {1,1,4,8,16}; //Exemplu de val. ale obiectelor
        int capacity =10 ; // capacitatea rucsacului;
        backpack (values, capacity);
        return 0;
    }
```