INS Practical 2

Date: **09-02-2023**

Roll no.: **20BCE119**

Name: **Kartavya Patel**

Course Code and Name: 2CSDE54 **Information and Network Security**

Task

- Implementation of Transposition ciphers (Single as well as Multilevel)

```
const cleanPlaintext = (pt) => {
  return pt.split(" ").join("").toUpperCase();
};

const encryption$1 = (pt, depth) => {
  pt = cleanPlaintext(pt);
  if (pt.length === 1) return pt;
  if (depth === 1 || !(pt.length / depth > 1)) {
    throw Error(
      `Depth must be in range of [2, ${pt.length - 1}] in this case.`
    );
  }
  let row = 1,
    rowStep = 1;
  let matrix = {};
  for (let index = 0; index < pt.length; index++) {
    if (matrix[row] === undefined) matrix[row] = "";
    matrix[row] += pt[index];
    if (row === depth) rowStep = -1;
    else if (row === 1) rowStep = 1;
    row += rowStep;
```

```
  }
  let encrypted = "";
  Object.keys(matrix).forEach((element) => {
    encrypted += matrix[element];
  });
  return encrypted;
};
const decryption$1 = (ct, depth) => {
  if (depth === 1 || !(ct.length / depth > 1)) {
    throw Error(
      `Depth must be in range of [2, ${ct.length - 1}] in this case.`
    );
  }
  let couter = 0,
    matrix = {};
  for (let depths = 1; depths <= depth; depths++) {
    let row = 1,
      rowStep = 1;
    matrix[depths] = "";
    for (let index = 0; index < ct.length; index++) {
      if (row === depths) {
        matrix[depths] += ct[couter];
        couter++;
      }
      if (row === depth) rowStep = -1;
      else if (row === 1) rowStep = 1;
      row += rowStep;
    }
  }
  let decrypted = "",
    row = 1,
    rowStep = 1;
  for (let index = 0; index < ct.length; index++) {
    decrypted += matrix[row][0];
    matrix[row] = matrix[row].substring(1);
    if (row === depth) rowStep = -1;
    else if (row === 1) rowStep = 1;
```

```
    row += rowStep;
  }
  return decrypted;
};

const encryption = (pt, encryptionKey) => {
  pt = cleanPlaintext(pt);
  let key = encryptionKey.toString(),
    cols = key.length,
    rows = Math.ceil(pt.length / cols),
    extras = rows * cols - pt.length - 1,
    matrix = {},
    encrypted = "";
  if (cols <= 1) throw Error("Key must be at least 2.");
  for (let index = 0; index < rows * cols; index++) {
    if (matrix[key[index % cols]] === undefined) {
      matrix[key[index % cols]] = "";
    }
    if (pt[index] === undefined) {
      matrix[key[index % cols]] += String.fromCharCode(90 - extras);
      extras--;
    } else {
      matrix[key[index % cols]] += pt[index];
    }
  }
  Object.keys(matrix).forEach((element) => {
    encrypted += matrix[element];
  });
  return encrypted;
};
const decryption = (ct, encryptionKey) => {
  let key = encryptionKey.toString(),
    cols = key.length,
    rows = ct.length / cols,
    decrypted = "";
  if (cols <= 1) throw Error("Key must be at least 2.");
  for (let row = 0; row < rows; row++) {
```

```
        for (let col = 0; col < cols; col++) {
            decrypted += ct[(Number(key[col]) - 1) * rows + row];
        }
    }
    return decrypted;
};

const transposition_ciphers = {
    rectangular: {
        encryption: encryption,
        decryption: decryption,
    },
    rail_fence: {
        encryption: encryption$1,
        decryption: decryption$1,
    },
};
const pt = "Information and Network Security",
    rail_fence_depth = 3,
    rectangular_transposition_key = 32541;
const renctangular_encrypted_text =
        transposition_ciphers.rectangular.encryption(
            pt,
            rectangular_transposition_key
        ),
    rail_fence_encrypted_text = transposition_ciphers.rail_fence.encryption(
        pt,
        rail_fence_depth
    ),
    test = {};
test[`Rectangular transposition (key=${rectangular_transposition_key})`] = {
    Encrypted: renctangular_encrypted_text,
    Decrypted: transposition_ciphers.rectangular.decryption(
        renctangular_encrypted_text,
        rectangular_transposition_key
    ),
};
```

```
test[`Rail fence (depth=${rail_fence_depth})`] = {
    Encrypted: rail_fence_encrypted_text,
    Decrypted: transposition_ciphers.rail_fence.decryption(
        rail_fence_encrypted_text,
        rail_fence_depth
    ),
};
console.log(`Plain text: "${pt}"`);
console.table(test);
```

Output

```
kp@KPs-MBP prac2 % node 20BCE119_INS_prac2.js
Plain text: "Information and Network Security"
```

(index)	Encrypted	Decrypted
Rectangular transposition (key=32541)	'RONRUZNAATSIIMNEKROIDOCYFTNWET'	'INFORMATIONANDNETWORKSECURITYZ'
Rail fence (depth=3)	'IRINTKUYNOMTOADEWRSCRTFANNOEI'	'INFORMATIONANDNETWORKSECURITY'